Diagnosing Performance Problems with WebSphere Application Server on z/OS

H. Michael Everett – WebSphere Integration Test

There is always a need to be able to diagnose performance problems. In order to diagnose performance problems on any combination of machines, with any combination of products, there is a certain amount of conceptual material to introduce. The information in this paper, while using WebSphere on z/OS as an example, is applicable to any machine running a J2EE application server. This paper is divided into four parts.

1. Questions and answers to introduce conceptual material
2. A flow chart of places to search for answers to your performance problem
3. Some real examples of what was done for various performance problems
4. A list of hints and tips that will potentially eliminate performance problems before they occur

Introduction

1. Questions and answers to introduce conceptual material
Constantly asking questions is the only way to understand what is involved in owning your own J2EE environment. This document should help define the questions to ask. Once the questions introduce the concepts at a high level, a flow is determined from the user to the data. It is a “religious” discussion as to whether to start with the data or the user. This paper starts with the user because the document is designed for consumption by IBM customers. IBM customers start with what they can see and diagnose inward. If this paper were designed for IBM support personnel, it would begin at the data or internal to WebSphere because support often starts with a console dump and diagnoses outward. This is an important point because it explains many communication problems that occur between people with a problem and whatever support they have to call upon. The different sides are starting from different perspectives; each understands their own point of view but don’t often consider the other’s viewpoint.

2. A flow chart of places to search for answers to your performance problem
The flow chart provides a means of narrowing down where a problem may be located. The combination of the concepts and diagnosis flow will imply a general means of solving any problem. If you understand how something works and you understand the flow from part to part, at a very minimum you will be able to find out where your problem is occurring, which is 90% of the battle.

3. Some real customer examples of what was done for various performance problems
The examples are in the third section of this document. There are a number of symptoms and causes discussed in the examples. All of the examples come from actual customers. The symptoms and causes are listed below.

The **symptoms** for performance problems come in several categories:
- I am not getting enough requests per second
- I am using far too many resources and cannot afford to pay for this solution
- I have a laptop that runs the application faster than my expensive server
- I didn’t change anything and performance is worse
- A combination of any of these

The **causes** also come in a surprisingly limited number of categories:
- The amount of instructions the processor has to process is simply too many. This is called path length and we want it to be shorter
- A request for some type of resource cannot be supplied
  - Processing has stopped because something is waiting for an answer or a resource that is busy
  - The hardware has hit its maximum and no more is available
  - Some decision making software is limiting resources on the systems behalf

Below is a quick reference of the Customer Examples:
- **Example 1:** Increasing DB2 and MQ background threads fixes slow response time
  "I am expecting 20 Transactions per second and I am only seeing two."
- **Example 2:** Adjusting the Workload Management Policy increases throughput to expected levels
  "We expect a minimum of 100 transactions per second and are only seeing 85."
- **Example 3:** An IP change and an application XML change reduced response time
  "The customer presses the save button and the page doesn’t come back for 20 seconds."
- **Example 4:** An upgraded level of the JVM and many other small changes solved this problem
  "Help us run 90% of our workload in less than 1 second, get 125 transactions per second, and, oh yeah, not use to much CPU"
- **Example 5:** changing the application architecture solved this problem
  "We are seeing 1.7 transactions per second and want to see at least 6"

4. **A list of hints and tips that will potentially eliminate performance problems before they occur**
This entire paper is meant to be a living, changing document. This list started out as a top 10 list and at 15 items has become, simply, hints and tips.
The current list of Customer Examples

Example 1: Increasing DB2 and MQ background threads fixes slow response time
   “I am expecting 20 Transactions per second and I am only seeing two.”

Example 2: Adjusting the Workload Management Policy increases throughput to expected levels
   “We expect a minimum of 100 transactions per second and are only seeing 85.”

Example 3: An IP change and an application XML change reduced response time
   “The customer presses the save button and the page doesn’t come back for 20 seconds.”

Example 4: An upgraded level of the JVM and many other small changes solved this problem
   “Help us run 90% of our workload in less than 1 second, get 125 transactions per second, and, oh yeah, not use to much CPU”

Example 5: changing the application architecture solved this problem
   “We are seeing 1.7 transactions per second and want to see at least 6”

Example 6: turning off trace reduced MIPS to a satisfactory level
   "We expect 1.3 MIPS per transaction and we are seeing 8.7"
1. Questions and answers to introduce conceptual material

All of these questions are answered in terms of items needed to diagnose the performance problem. In other words, the concepts are explained for the purposes of this paper but not necessarily inclusive.

How does a computer actually work?

Knowing the basic operation of a computer is absolutely fundamental to solving performance problems. It does not matter whether it is a mainframe or laptop; all computers basically work the same. With this in mind, we ask the question: When you buy a personal computer or laptop, what do the advertisers make sure you know? They make sure you know the CPU speed in Gigahertz.

**CPU Speed**

For example, my computer has a Centrino Intel processor running at 1.59GHz. This matters to you because there is a clock that determines how fast a computer can process one instruction. The clock lives inside or is the speed of the CPU. How fast your clock ticks determines how fast you can process that single instruction. A single mainframe CPU may run at 750MHz while your new laptop may run at 2.4GHz. If you compare a single CPU test with a CPU intensive application, the laptop will win but you have missed the point of scaling up a mainframe with multiple processors and thousands of concurrent users.

**Number of Processors**

Implied in the previous section is that a single processor processes one instruction at a time. For the most part this is true, though there are some new processors in the market that have multiple cores that will execute multiple instructions simultaneously, but they are not applicable to this paper. So how do I speed up processing of instructions? I get a faster clock or I add more CPUs. Each CPU can process one instruction at a time. Speeding the processing of instructions, however, is only one way to increase performance. We often reduce the number of instructions (or path length) so the end result is the same with less instructions for the processor to process. Don’t forget these two points: scaling or increasing the number of CPU’s and reducing the path length or number of instructions are fundamental to performance tuning.

What else do advertisers make sure you see?

**Amount of memory**

Let’s say a computer has 1.5 Gigabits of RAM. RAM (Random Access Memory) is a very fast temporary storage space for the instruction waiting to be processed by the CPU. It has a finite size. The more instructions you try to run on your computer, the more you are trying to place in the RAM, waiting to be processed. If the RAM fills up and you need to execute additional code, code in RAM is moved to a special file on the hard drive or disk to make room for what you want to be processed. This movement to the hard drive is called paging. Paging happens to be one of the slowest things you can ever do on any computer. This is a simplified view, but the idea is clear. Another name for this process is I/O or Input/Output. I/O makes the computer use mechanical parts, the read/write heads have to physically move in the hard drive. This is very slow. As a result, with performance in mind, we want to do as little I/O as possible. I/O is stated to be one of the slowest because waiting and doing nothing is even slower. Not to be facetious, but waiting is a big cause of performance problems. If one step has to complete for the next to occur and the first takes a long time, no progress is being made while the time consuming process is occurring.
Disk Space
Advertisers also mention the size of the hard drive. The size of the hard drive, although interesting, is not relevant to this paper. We will assume you have the products you need already installed on available disk space. Remember paging is bad, I/O is bad, so monitor the amount of space to which your computer pages, make sure there is enough space to page into when paging is necessary and make sure paging is at a minimum.

How does z/OS actually work?
z/OS is the current state of evolution of a 40-years-young IBM mainframe operating system. It is robust, proven, stable, and capable of processing any business you want it to; however, it is unlike anything you are used to seeing at home. The actual mainframe (hardware) is called zSeries and comes in many different models with varying numbers of CPUs and memory.

Documentation for z/OS can be found at the following URL:

Documentation comparing various models of zSeries can be found at the following URL:

This is important to you because, unlike a laptop with Windows or Linux, manual intervention is necessary to set up TCP/IP, UNIX, database products, etc. On the mainframe there are executable files and textual parameter files. Files are called datasets in mainframe-speak. At machine startup, the parameter datasets are used to determine how much resource the machine allocates for various products. A simplified view but enough to get us started. It is important to remember that everything you need is an individual part of the operating system, or an individual product that will have its own parameters needing manual attention. Most parameters will be fine with the default values; others will be mentioned throughout this paper as items to remember. If you are not working on a mainframe, you will need to discover where parameters are set for the various products you wish to use.

How does WebSphere on z/OS actually work?
WebSphere on z/OS is taken from tape cartridges or downloaded and placed into datasets. A text-based series of panels allows you to configure WebSphere on the mainframe. Hundreds of items specific to your mainframe system are input into the panels with an end result of jobs and instructions. Using those instructions, system personnel will manually edit system parameters in specified datasets and then execute the jobs to configure WebSphere. Why is it important to mention this scenario? Most other platforms have some type of automated installation program to guide the user. WebSphere on the mainframe is probably the easiest to misconfigure because it requires manual editing of numerous parameters. Even with guided installations manual intervention is usually required. For instance, you will have to enter a host name during any installation of any J2EE server on any platform. What if that hostname you entered is configured to go through a particularly slow router or along a path in the network that is 100Mb rather than Gigabyte transfer speeds for some reason? Only with help from your network personnel could you know, and only after eliminating many other items would you look at the network. Hopefully, the flow described within this document will help eliminate some errors in manual configuration.

Three important documents exist that introduce WebSphere on z/OS.

  Introduction to WebSphere V5:
  http://www-03.ibm.com/support/techdocs/atsmastr.nsf/WebIndex/WP100339

  Planning Production and Test:
  http://www-03.ibm.com/support/techdocs/atsmastr.nsf/WebIndex/WP100396
Within the z/OS operating system’s base components, WebSphere absolutely needs TCP/IP, a Unix environment (Unix Systems Services), a transactional engine (Resource Recovery Services), Logging Facilities (Logger), Workload management (WLM), security (Resource Access Control facility (RACF) or another Security Authorization Facility (SAF) equivalent security product) and Sysplex (a means of coupling systems together with fast communication capability). This list does not mention the work the operating system kernel does pushing and pulling work through the hardware.

What subsystems do you interoperate with on z/OS and outside z/OS?

We have mentioned the hardware (memory and CPU), the operating system software, and WebSphere installed on the operating system. What about the other software products with which WebSphere interacts? So far we have not mentioned any data. A computer’s ultimate purpose is to manipulate data. In that respect there will definitely be at least one database product and maybe many products that perform business logic or push the data around. You will need to know what other products are in use, how to configure them, and how they interact with your J2EE server.

The following paper demonstrates the recommended best practice for the common products and WebSphere on z/OS (benchmark data included):


The common products that fall into this category for z/OS are:

- **DB2** – relational database engine for storing data
- **WebSphere MQ** – messaging product for moving data
- **CICS** – transaction engine for manipulating and moving data
- **IMS** – database engine for storing data
- **Another J2EE server** – transaction engine for manipulating and moving data
- **Standalone program** – could be doing anything

You may be using all or none of these. The database may live on a system that is across the world from your WebSphere environment. The important items to discover are: how is my application server interacting with the other product; where to alter runtime parameters for these other products, and where the other products live.

How many different machines are talking back and forth?

In the coming sections we will describe making a flow from the user to the data. In order to make this flow you need to know how many machines are involved in your critical business application. We can expand that to how many machines, what products are on each machine, and how many users are trying to use them at various points in the day.

Often the only symptom that is given is: “it takes longer for my web page to display” or “my web page does not display at all.” Which leads to the question: “what page are you accessing and what were you doing at the time?”

If the user was logging on, we would need to know whether there is an external machine that handles authentication for something like single sign-on to know whether we need to look there or whether we can skip it. If the user is past the logon, we can move further into the flow, past the machine doing authentication. Suppose
the user is requesting a function that accesses CICS to gather user information. We need to look and see if the CICS regions are running, how many there are, what machines they live on, and how WebSphere interacts with CICS. Upon gathering these items we can look for what might need to change to alleviate the problem.

**Items to remember:**
We want to either reduce path length on the poorly performing machine or find which product is waiting for what resource.

**Hardware and software items of interest:**
- How many machines;
- How many CPUs on each machine;
- How much memory on each machine;
- What software, including version numbers, is relevant on each machine?

**How do I diagnose performance problems with WebSphere on z/OS?**

**Ask a bunch of questions:**
1. Have you performed the basic WebSphere tuning outlined in the InfoCenter?
   There are four layers of tuning on z/OS:
   - Tuning the z/OS operating system
   - Tuning for subsystems
   - Tuning the WebSphere Application Server for z/OS runtime
   - Tuning for J2EE applications

   There are six layers of tuning on other platforms many of which will also benefit z/OS:
   - Tuning application servers
   - Tuning Java virtual machines
   - Tuning applications
   - Tuning databases
   - Tuning Java messaging service
   - Tuning security

2. Follow these general guidelines for z/OS
   (These guidelines are presented in greater detail in the following presentation:
   http://www-1.ibm.com/support/docview.wss?uid=tss1prs804&aid=1)
   - Paging is bad; give the address spaces enough memory. There are three places to limit address space memory: the started procedure JCL; an IEFUSI exit; Unix Systems Services parameter maxassize.
   - Use the coupling facility for all logging
   - Make sure your DASD has caching turned on
   - Use Gigabit network connections with QDIO
   - Put executable members into central storage. This means there will be only one copy instead of using storage to copy the members into every address space.
   - For USS make sure the following parameters are high enough:
     - Update BPXPRMxx parmlib member
• MAXFILEPROC the maximum number of files that a single process can have concurrently active or allocated. Values can range from 3 to 65535. The default is 64. The IBM performance test system is set to 5555. Set MAXFILEPROC high enough - impacts OMVS kernel storage, only set as high as needed, applies to all USS user processes (or set at user level using RACF)
• MAXSOCKETS parameter on AF_UNIX & AF_INET NETWORK statements the maximum number of sockets supported by this socket filesystem. Values can range from 0 to 64498. Set MAXSOCKETS for the AF_INET filesystem high enough, at least as high as MAXFILEPROC (No Impact on OMVS kernel storage)
• MAXPROCSYS the maximum number of OS/390 UNIX processes that the system allows. Values can range from 5 to 32767. The default is 200. The IBM performance test system is set to 900.
• MAXPROCUSER the maximum number of processes that a single OS/390 UNIX user ID can have concurrently active, regardless of how the processes were created. Values can range from 3 to 32767. The default is 25. The IBM performance test system is set to 32767.
• MAXTHREADTASKS the maximum number of MVS tasks that a single process can have concurrently active for pthread_created threads. Values can range from 0 to 32768. The default is 50. The IBM performance test system is set to 9000.
• MAXTHREADS the maximum number of pthread_created threads, including running, queued, and exited but undetached, that a single process can have concurrently active. Values can range from 0 to 100000. The default is 200. The IBM performance test system is set to 90000.
• MAXASSIZE the address space region size. Values can range from 10485750 (10M) to 2147483647 (2G). The default is 41943030(40M). The IBM performance test system is set to 2147483647.
• MAXCPUTIME the CPU time, in seconds, that a process can use. Values can range from 7 to 2147483647. The default is 1000. The IBM performance test system is set to 9999.
• MAXSHAREPAGES the maximum amount of shared system storage pages that OS/390 UNIX functions can use. Values can range from 0 to 32768000. The default is 100. The IBM performance test system is set to 131072.
• VMAX the maximum virtual storage limit for HFS data and metadata buffers. The default is half the physical memory. The IBM performance test system is set to the default.

• Mount the WebSphere product HFS read/only
• Use SMF 92 records to tune file caching for HFS activity
• Use zFS for better performance
• Write log4j logstreams to unshared zFS, write simple trace strings, write without flush
• For enabled SAF classes, number of profiles in class will affect performance
• Define the BPX.SAFFASTPATH facility class. Don't use if you need to audit successful HFS accesses, or use the IRRSXT00 exit
• Security performance depends on your Repository Mechanism: Custom can be better than RACF, which is better than LDAP
• Workload profile determines the number of threads in a servant address space: Servers >> Application Servers >> server_name >> ORB Service >> Advanced Settings >> ’Workload Profile’ Test with different numbers of threads to determine the optimal setting
• Keep current with the latest Java SDK
• Run with JIT enabled
Investigate `-Xquickstart' & `-Xclass:noverify' (`-Xverify:none' in 1.4.1)

Other Java tips: http://www.ibm.com/servers/eserver/zseries/software/java

Eliminate the RRS archive log

3) No matter the problem in your production environment, start by reducing path length. We can easily reduce path length by turning off certain items:

- Turn off WebSphere trace
- Turn off application trace
- Turn off unnecessary trace for all z/OS subsystems (RACF, DB2, JDBC, etc.)
- Run minimal trace for monitoring tools (SMF 120 interval at 10 minute interval, do not use activity trace)
- Make sure parameters within WebSphere are minimally chatty
- Auto reload turned off: enterprise applications > application name: uncheck autoreload or check autoreload and set the interval to zero
- Use a local JDBC driver
- Use bindings mode for MQSeries
- All resource subsystems should be on the same machine as the application server whenever possible, limit going over the network in favor of local communication
- Use “pass by reference” in the ORB settings in the WebSphere Administrative Console: server > ORB settings > check pass by reference
- Set your min and max Java Heap to the same value. At server startup the JVM grabs the max, there is no reason to spend path length allowing the JVM to try and keep the heap at the minimum
- JAVA_MMAP_MAXSIZE=nn Max-sized jar files (in MB) that will be memory mapped. The default is 0; that is, memory mapping is not used. Jar files 'n' MB or smaller will be accessed using a memory mapped file
- Check the job output for exceptions. Exception processing is very expensive. Eliminate errors from the server output

4) Draw the flow through the application (see next section)

2. A flow of places to search for answers to your performance problem

We know there is a problem. What is wrong here? After this, the first question to ask is what levels of code are running for the products in the application flow? But we cannot answer that until we know the relevant products and where they live.

Start drawing a flow and adding the level of code to the picture we are making. The flow is described throughout this section.

The best thing to do is have this picture ready and updated as your production environment is built and grows. That way, when you encounter a performance problem, you do not have to spend a great deal of time finding the personnel involved in each piece to figure out what levels are installed on which machine. A change management process should be in place to record this data as the environment changes. If the data is recorded, the picture can be updated without involving too many people.
Create a picture or flow from the user to the data

Start

Thicker Client(s)
  yes
  Thicker Clients - What product? What Version? How many? Where are they? Expected throughput per second at various periods of the day? Is the browser able to ping the next machine it needs? What protocol do the clients use?
  no

Batch Jobs - What product? What Version? How Many? Where are they? Expected throughput? How long do the automatic jobs run? Are they running during peak hours, if so can they run at a low peak time frame?

Web Browsers - What product? What Version? How many? Where are they? Expected throughput per second at various periods of the day? Is the browser able to ping the next machine it needs? Are you pulling the page from cache in the browser? Does your app create cookies? How big is the session data?

Where does the application flow start?

Edge/Blade server(s)
  yes
  Some type of caching server(s) - What product? What Version? How Many? Where are they? What are you caching (static, non-static, both), What parameters are available for changing?
  no

next

Firewall
  yes
  Firewall - What product? What Version? How Many? Where are they? Are there any trace settings on by default. Are the necessary ports open? Are blocked ports causing expensive getHostByName failures and the like? Do you think the application made it past this step?
  no

next

What are the intermediate hops?
Intermediate hops continued

Getting to your application server

Getting to your data

Monitoring tool data Collector – What product? What Version? How Many? Where are they? What parameters are available for changing? How much data is being collected? Can the amount of data be lowered? Do you think the application made it past this step?

HTTP Servers – What products? What Version? How Many? Where are they? Are there any trace settings on by default. Are you using FRCG cache or available caching? Are the servers in a scalable mode or simply sharing ports? Are there enough servers to handle incoming load? What is the expected load at various times of day? Are you using a plugin to route work to an appserver? Do you think the application made it past this step?

Access Manager - What products? What Version? How Many? Where are they? Are there any trace settings on by default. Are you using a third party authentication method? How is it accomplished? Where is SSL used? Where is the data store for user information (on this machine or another) Do you think the application made it past this step?

HTTP/Protocol Sprayer – What product? What Version? How Many? Where are they? Are there any trace settings on by default. How does the product determine where to send a request? Is the product decision process configurable? What is the expected load at various times of day? Do you think the application made it past this step?

Is there anything else in the middle that we have not covered like a CORBA server, etc...

Application Server(s) – What product? What Version? How Many? Where are they? Are there any trace settings on by default. Do you think the application is making it this far?

Data Resource Managers – What product? What Version? How Many? Where are they? Are there any trace settings on by default. Do you think the application is making it this far?

DASD – do you have enough page DASD? Is caching turned on within your DASD hardware? Old DASD is slower than newer DASD.
3. Some examples of what to do when you find the location of a performance problem

Hopefully, while working through the flow of your application, you have a general idea of where different products interact. We need to drill down further into many of the topics in the flow chart with examples, but mainly the Application server and Data Resource Manager pieces of the flow. The rest of the paper will detail real customer experiences, explaining the original symptom and how we were able to alleviate the performance woes.

IF YOU HAVE NOT FOLLOWED THE BASIC TUNING INFORMATION FROM THE WEBSHERE INFORMATION CENTER, DO SO NOW. IT IS NOT A SUGGESTION. THE ITEMS LISTED ARE ESSENTIAL TO SUCCESS.

Pointers to the basic Tuning Information are detailed in the first section of this document. The reason we insist on performing steps in the Information Center is to help eliminate the runtime and application server as a cause. Eliminate the runtime and application server and it is time to look into what the application is doing.

A growing section of Customer Experiences:

Example 1: Increasing DB2 and MQ background threads fixes slow response time
“I am expecting 20 Transactions per second and I am only seeing two.”

We followed the suggested tuning in the InfoCenter, summarized in the following cheat sheet: http://www-1.ibm.com/support/docview.wss?uid=swg27005039&aid=1.
The cheat sheet is not a replacement for the InfoCenter; it is provided as a reminder so all topics are in one place for viewing.

In this situation there were no real symptoms given other than things are slow. We began to ask the series of questions in the flow chart. While doing this we determined the following:
  - Single system testing a new application
  - Newest version of WebSphere at the time
  - Appropriate levels of JDBC, WebSphere MQ, DB2, and Java SDK.
  - The application was using MDBs, communicating with WebSphere MQ, to manipulate data in DB2.
  - No clear reason for the slow response time yet.

At this point we spent two days looking at the environment, turning off unnecessary items, and no cause was clear. Performance was noticeably, but only slightly, better. We had not eliminated the runtime as the source of the problem, so a deep dive into the application was not the primary concern yet. Before blaming the application, a thorough analysis of the runtime and cooperating products is necessary. Next we needed some kind of a tool to dig deeper into the application flow. For this problem, a profiling tool was used to lead us to where the delay was occurring.

Throughout the examples, we will describe the tools and methods we use to diagnose performance problems. Let us introduce the first subset of tools available when you have no idea where to go:
  - Trace
Garbage Collection (GC) trace is the easiest and most useful tool to eliminate an overactive JVM as a first concrete symptom. Although this did not help us in this example, it is the first tool mentioned because everyone should check GC on a regular basis as routine performance maintenance.

When the JVM starts up, it grabs a certain amount of memory based on two environment variables named with some variation of heap size and max heap size. As Java code executes it fills up the heap. When a piece of Java code requests storage and the heap does not have the storage an allocation failure occurs. The allocation failure causes a process called garbage collection to go through the heap and claim the space taken by unnecessary objects. While garbage collection runs, all other work in the Java Virtual Machine (JVM) stops. Notice that if GC is running, nothing else can or will. For this reason, keep GC execution at 5% or less of the total executing time. If, from the trace, GC is occurring more than 5% of the total time, investigate how to reduce it. Look for items in the application code that may be requesting the largest chunks of memory.

Documentation on the z/OS JVM diagnosis can be found at the following URL:

IBM Support personnel may ask for any number of trace settings and output

- Monitoring tools
  - Tivoli Performance Viewer (TPV) is shipped with the WebSphere Application Server. TPV uses a WebSphere monitoring tool called Performance Monitoring Infrastructure (PMI). When enabled PMI produces minimally invasive trace for tools to read and display. PMI can be used in a production environment when set at its low levels. TPV is a client that connects to the application server using the SOAP or IIOP protocols. With this performance problem the performance adviser option built into TPV advised us to increase the connection pools for our J2EE resource connecting to DB2. We did so and, although this helped in the long run, it did not solve our performance problem.

  Documentation for Monitoring Overall System Health can be found at the following URL:

  Documentation for TPV can be found at the following URL:

  Documentation for PMI can be found at the following URL:

- Profiling tools
  - Profiling tools hook directly to the JVM to record all, or subsets of all, activity occurring within the JVM. There is a profiling tool shipped within WebSphere Studio Application Developer (WSAD), also Rational Application Developer (RAD). Profiling tools are different than monitoring tools in that they are extremely invasive. You will not be able to profile an application while in production due to the excessive overhead of the tool. We used a version of the profiler shipped with WSAD to lead us to the answer for this example problem.
Profiling tools are a development time tool that emergency support personnel use to quickly see everything that is happening in the JVM. Depending on how the profiling tool displays the data, a user can quickly and visually see which methods are taking the longest amount of time to complete. When we looked at this data for example 1, we noticed the native JDBC driver code was executing in seconds. JDBC code should always be running in the millisecond or microsecond timeframe as a rule.

Documentation for profiling in WebSphere Studio Application Developer (WSAD) can be found at the following URLs:


Because of what was in the profiler trace, the next series of questions happened to include how many threads are running in the DB2 engine. The number of threads was at the default of 20 for zSeries DB2. WebSphere on z/OS typically requires more than 200 depending on the application and its interaction with DB2. Background threads set at 300 would be common on z/Series DB2. Upon increasing the ZPARM value IDBACK to 300, our performance was immediately at an acceptable level for the customer.

This example is first in the paper because this particular performance problem is hard to see. IBM has not done a good job of displaying when internal threads to DB2 and WebSphere MQ are set to low. Since the customer application was also using WebSphere MQ, we checked to make sure the IDBACK setting in WebSphere MQ was also set higher than the default. It was set at 400, thus no change was necessary.

**Example 2: Adjusting the Workload Management Policy increases throughput to expected levels**

“We expect a minimum of 100 transactions per second and are only seeing 85.”

IBM support personnel and the customer began by drawing a picture of the environment with product code levels in the respective locations. No glaring problems were found. All code was as current as possible. No clear idea of where the bottleneck is located was obvious. In a situation where there is no obvious direction IBM personnel in a mainframe environment will typically look at the Workload Management (WLM) policy. The reason to start with WLM is because it is the only product that captures the goals of all products in the mainframe environment. From the initial picture, WebSphere executed some business logic that interacted with an IMS database. Because of this the WLM policy was checked for these two products

We made 2 changes in this situation:

- We changed some WLM definitions to get to 100 trans/sec
  - Adjusted the WLM service class for WebSphere to 90% of transaction finish in .5 seconds, they were set at 99% of the work in .3 seconds.
  - If you are too aggressive with WLM, WLM will decide to ignore your request in favor of the rest of the system. If it were to try and satisfy 99% of the work in .3 seconds many other items on the system would suffer. At 90% in .5 seconds we achieved the desired goals
If the service class goal exceeds 90% of the work or asks for the work to complete in less than .5 seconds the goal may be to aggressive. There is no sure way to notice if WLM has bypassed your aggressive goal in favor of the rest of the system. A rule of thumb for noticing if the WLM goals are to aggressive is to watch how much CPU is being used. During production loads if the application is not using the maximum amount of CPU no matter how many users are pushed into the system, the goal may be to aggressive. If WLM is trying to meet the aggressive goals the resources on the system will be pushed to the limits, especially on a constrained system.

The recommended information for WLM configuration is at the following URL:

- We implemented even distribution over servant regions.
  - Set the WebSphere environment variable wlm_stateful_session_placement_on=1
  - Details for even distribution are in the InfoCenter at:

It is important to remember that J2EE applications rely on the application server and other products to manipulate data. The application server isn’t very interesting if it does not interact with data. It is the J2EE resources defined that allow application servers to interact with products like databases. If there is a product like WLM that controls system resources for many products a decision has to be made as to the relative importance of each product it controls. We suggest giving WebSphere an aggressive goal and the products it depends on a slightly more aggressive or equal goal. The URL for WLM configuration with respect to WebSphere is listed two paragraphs above.

Example 3: An IP change and an application XML change reduced response time
“The customer presses the save button and the page doesn’t come back for 20 seconds.”

We made 1 configuration change and 1 application change to go from over 17 seconds per save to 1 second per save.

- We profiled the application and found that a third party JDBC driver was taking 12 of 17 seconds doing a getHostByAddress resulting in an unknown host exception. The customer took a packet trace and discovered a mistake in Virtual IP Address (VIPA) setup.
  - A tool that can be used to analyze a packet trace can be found at the following URL:
    http://www.ethereal.com. The Ethereal product has an Open Source license. Please review all licensing if it is the product you choose to analyze packet traces.
- From the profiler trace we could see that the application was not caching the transformation and was spending a great deal of time compiling during XML processing.
  - We went to the following IBM XML Performance Web site and found the answer that worked:
    http://www.ibm.com/servers/eserver/zseries/software/xml/performance. Navigating this site we found that we should be caching the Template used in XSLT processing and how to do so. You can cache the Templates object and create new transformer from the Templates object or you can cache the transformer object directly. The Transformer object is NOT thread safe, on the other hand, the Templates object IS thread safe. One copy of the Templates object is reused by all the threads (to
create their own transformer). It is relatively inexpensive (compared with the actual transformation) to create a transformer from the Templates object.

```java
public static synchronized Templates getTemplates(String fileName) throws javax.xml.transform.TransformerException {
    if (cachedTemplate != null) {
        return cachedTemplate;
    }

    TransformerFactory tFactory = TransformerFactory.newInstance();
    //TransformerFactory is //not guaranteed thread safe
    cachedTemplate = tFactory.newTemplates(new StreamSource(fileName));
    return cachedTemplate;
}

Templates template = getTemplates(xsl);
Transformer transformer = template.newTransformer();
....do your transformation....
```

**Example 4: An upgraded level of the JVM and many other small changes solved this problem**

“Help us run 90% of our workload in less than 1 second, get 125 transactions per second, without using too much CPU”

The customer was running 6 servant address spaces on 2 LPARS with 512M heap and 20 threads per servant. The repeatable test was created using the LoadRunner product. Load Runner is from a company called Mercury Interactive and can be found at the following URL: [http://www.mercury.com/us/products/performance-center/loadrunner/](http://www.mercury.com/us/products/performance-center/loadrunner/)

**Changes to the system other than those mentioned in the basic tuning section and items we needed to verify**

- On the bind for the universal JDBC driver (JCC) - specify cache dynamic=yes and need to rebind JCC with keepdynamic=yes which means the application does not need to explicitly re-prepare the statement after a commit operation. This saves path length at the slight risk of witnessing errors upon commit. This parameter is described in the documentation pointer below. Also, set a property in WebSphere to make thread level caching work: keepDynamic=1 as a datasource custom property. This was an item we simply knew to check. Upon looking for it in the documentation, we realized it was not easy to find. KeepDynamic was not found searching the WebSphere documentation at the time of writing this document.
  - The following section of the WebSphere Information Center describes of Tuning DB2 for z/OS:
  - The DB2 documentation page is located at the following URL:
  - We found the keepDynamic information at the following URL:
- Put the log4j logs into their own zFS. There was no way to change to no flush, the log4j implementation was hard coded. If using another logging method, make sure it is set to minimal levels.
  - zFS documentation can be found at the following URL:
log4j documentation can be found at the following URL:
- [http://www-03.ibm.com/support/techdocs/atsmastr.nsf/032f6e163324983085256b79007f5aec/5a3fc63d27a9d8cc86256dbf0075dbb4?OpenDocument](http://www-03.ibm.com/support/techdocs/atsmastr.nsf/032f6e163324983085256b79007f5aec/5a3fc63d27a9d8cc86256dbf0075dbb4?OpenDocument)

- The logger logstreams did go to coupling facility. We did not have to eliminate the RRS archive log.
- RACF was not auditing any accesses, thus no changes were necessary for this item.
- In the DB2 ZPARMS they had CACHEDYN=Y but had MAXKEEPD=0, increased MAXKEEPD=10000. This information is discussed in the section of the WebSphere Information Center discussing Tuning DB2 for z/OS mentioned a few bullets above.

Although these items improved performance, the change producing the results we were looking for was upgrading the level of the Java Virtual Machine (JVM). The customers JVM was four maintenance levels behind.
- GC was very strange. There was a GC every 1/3 second with no allocation failures. Changed JDK to the latest level resolving an old issue with zip file processing in which it called System.gc() often.

**Example 5: changing the application architecture solved this problem**

“We are seeing 1.7 transactions per second and want to see at least 6”

Most performance problems are solved by changing the application, IBM Support personnel will usually eliminate the runtime even if the instinct at hand is to dig into the application. Most problems are fixed by digging into the application. That said, IBM and this customer spent a few weeks trying to see where the bottleneck was in this particular application. We did this because the customer was not receptive to digging into their code until we could prove the problem was in the application.

The flow through the application was like so: There is no end-user front-end. Information about the employee is passed from a device onto WebSphere MQ Series Queues, Message Driven Beans (MDB) in the application server use Plain Old Java Objects (POJO) to pull the message off of the queue and pass it to DB2 stored procedures using a data source and ultimately SQLJ. There is other processing but this is enough to describe the problem.

We used the following tools to try and see the bottleneck:
- WLM Policy for the system
- SMF 70-79 records to see the WebSphere workload activity
- SMF 177 and 233 to see DB2 stored procedure activity
- SMF 101 to see DB2 server activity
- An application profiler in the WebSphere server and Java Stored Procedure Address Space
- GC Trace in the WebSphere server and Java Stored Procedure Address Space
- Instrumented application code with the SMFJActivity class in the pmi.jar. This instrumentation is documented at the following URL: [http://www-1.ibm.com/support/docview.wss?uid=ts1td101339](http://www-1.ibm.com/support/docview.wss?uid=ts1td101339)
- Candle Omegamon for DB2 was being installed towards the end of our engagement

Documentation discussing how to configure and gather SMF data can be located at the following URLs:

Clue Number 1 – marshalling/un-marshalling
The first interesting thing we found was using the application profiler on the WebSphere JVM. The data showed that a large percentage of the time was being spent marshalling data. When a Java application wants to send an object over a network, it must first turn all necessary items within the object into a byte stream. The byte stream is sent over the network and the receiving party un-marshals the object or takes the byte stream and turns it back into an object. The same process happens again with the reply.

Many questions and several different tests were created to find out if this was our bottleneck. The application was making up to 45 stored procedures calls, each doing the marshalling/un-marshalling. Also, 5 of the 6 primary Java Stored procedures were performing single SQL statements. With such small amounts of processing and so many calls to these stored procedures the overhead in WebSphere and DB2 of creating and executing the calls must be a great deal more than executing the application code. As one member of the team looked into data marshalling (application), another searched for more bottlenecks in the infrastructure (system) including overhead of calling the stored procedures.

The instrumented SMFJActivity trace showed that the marshalling was a significant issue but we also wanted to prove that the overhead of calling the stored procedures with such little processing was a problem. We started looking at SMF data. In order to accurately look at SMF data for WebSphere you have to first look at the WLM Policy.

Clue Number 2 – the WLM Policy with inappropriate goals similar to Example 1 in this paper

In drawing a flow through the application we realized the customer was testing on a constrained system with 3 CPUs. This particular LPAR was weighted at only 40 percent. This means it could get at most 40 percent of 3 CPUs or about 1.25 CPUs if the system was busy.

In looking at the WLM Policy, the goals for WebSphere enclave work or work in the CB subsystem may be too aggressive. We would need to see some SMF 70-79 records to be sure. If WebSphere is classified correctly and we are trying to push as much work as possible, this constrained system should be using all of its maximum 1 and a quarter CPUs.

Workload CB - Component Broker
2 service classes are defined in this workload.
* Service Class CBFAST - WAS .5 second RT
  Base goal:
  CPU Critical flag: NO
  # Duration Imp Goal description
  1 - --------- - ----------------------------------------
  1 2 95% complete within 00:00:00.500
The WLM policy has 2 different major responsibilities: the most important being classifying workload goals and the second being granularity of reporting when gathering SMF 70-79 records. We had the customer separate the various WebSphere address spaces into different report classes and then gathered the data.

**Clue Number 3 – SMF 70-79 Records at 1 minute intervals during the repeatable test**

With SMF 70 through 79 records you can quickly see CPU usage, transactions per second, system paging, and whether WLM is meeting its goals for WebSphere enclave work. Figure 1.0 shows some SMF data from before making any changes and the explanation of what to quickly notice. In the sample SMF data shown, only a few items of interest are discussed, much more information exists; however, for this engagement the important items are described. Figure 2.0 is after we changed the WLM policy a bit and redesigned the application due to Marshalling/un-Marshalling and singleton stored procedure calls.
Figure 1.0 Sample SMF 70:79 records without any changes to the system and application

1 END/S
2 APPL% CP

Transaction rate – this is the number of transactions per second at 1.92
CPU percentage – currently using less than 1 CPU when we know we could
have 1.25 or so. Notice the CPU is at 94% when we could be at
125% or more if the system has extra resources.
3 PAGE-IN RATES
Paging – there is no paging in this example all values are zero

4 EXECUTION DELAYS %
Delays –

CPU – we are delayed and want more CPU 15.6% of the time, the current WLM policy is not allowing the system to be aggressive with WebSphere work. Other work on the system is using the resource. We can state this because the APPL% CP is only at 94% when it could have 125 according to how the LPARs are weighted, more if the system is not busy.
QMPL – there is a little bit of work sitting on the WLM queues, this will be the case when we are waiting for CPU
I/O – there is a small amount of I/O
UNKN – delays caused by other products outside the 70:79 records

5 GOAL: The goal of 90% of the work is not nearly being met at 16.2%. Moreover, at 1.85 transaction per second we are far below the goal of 6. If WLM encounters a goal that is feels is too aggressive, it will ignore it in favor of the rest of the system. If WLM were to try and meet the impossible goal it may starve the rest of the system for resources. With the current configuration, WLM is not attempting to make the goals set for WebSphere.

Change Number 1 – Lower the goal for WebSphere enclave work according to what we see in Figure 1. Since WebSphere is not getting all the CPU it could have, we want to make a change so WLM is fairly aggressive. When looking at the histogram in Figure 1 we choose to lower the goal to 70% of the work in 1 second. This change will keep WLM working on meeting an obtainable goal. This is a setting for this constrained system only. When adjusting these goals, look at RMF data from the zSeries machine being used.

Upon making this change we did see an increase in CPU usage. This proved to us that WLM was attempting to meet our goal.

Change Number 2 – Implement some of the stored procedure code directly in WebSphere
We noticed the marshalling/un-marshalling of objects was consuming much of the time in our transactions. Although we were unable to prove it with data in the time we had to try, it was concluded that the cost of creating and calling the stored procedure for these five stored procedures was 10 times that of running the code itself. While inspecting the Java Stored Procedures, we identified 5 of the 6 most often called that were executing singleton SQL and java code. A member of the team implemented this code inside WebSphere as JDBC calls.

The effect of doing this is to completely eliminate any of the marshalling for those 5 most often used Java Stored Procedures. Also, since up to 45 calls were made for each transaction, much of the overhead of setting up the stored procedure was also eliminated.

In Figure 2, the results of our changes show the application meeting the customer’s goal of 6 transactions per second. We did have to adjust the number of MDB’s in WebSphere and the number of threads in the servants for several tests to discover optimal values. Not enough MDB’s causes too little work on the WebSphere end to be produced. Not enough threads causes work to accumulate in the WLM work queues.
Descriptions of the results of our changes are below Figure 2.0

**Figure 2.0** Sample SMF 70:79 records after changes to the system and application

Transaction rate – this is the number of transactions per second at 6.65
2 APPL% CP

CPU percentage – currently using all the CPU we can get on this constrained system at 129%. WLM is working on our behalf to meet the goal we set.

3 PAGE-IN RATES

Paging – there is no paging in this example all values are zero

4 EXECUTION DELAYS%

Delays –

CPU – we are delayed and want more CPU 31.7% of the time. We advised the customer to lift the CPU restriction on the system to test how much throughput they could achieve on an unconstrained system to more closely resemble their production environment.

QMPL – there is a little bit of work sitting on the WLM queues. This will be the case when we are waiting for CPU.

5 GOAL: The goal of 70% of the work is not nearly being met at 23.8%. We are achieving the maximum CPU, thus reducing the WLM goal had the desired affect. We are surpassing our goal of 6 transactions per second and will leave the WLM goals alone.

Example 6: turning off trace reduced MIPS to a satisfactory level

"We expect 1.3 MIPS per transaction and we are seeing 8.7"

This was a very short engagement. As stated throughout the paper the first thing done in a situation where

4. A list of hints and tips that will potentially eliminate performance problems before they occur

The WebSphere V6 configuration cookbook can be found at the following URL:

1. Thread settings
   a. (WORKLOAD_PROFILE, etc). Compute the number of threads needed to support the peak transaction throughput rate.
   b. Workload Profile can be found at the following InfoCenter URL:

2. Number of servant regions.
   a. Find out the optimal number, you don't want a large number of threads going through a single servant region. If the servant region is taken down / crashes, recovery of the large number of transactions takes longer, for an online (web) application a large number of users may suffer a "glitch" / outage.
   b. Information on number of servant address spaces can be found at the following InfoCenter URL:

3. Connection Pool size.
   a. Check the deployment descriptors and verify if resource-references are being used. If resource-refs are in use, verify the Shareable / Non-Shareable tag.
   b. Information on connection pooling can be found at the following InfoCenter URL:
4. **Memory usage of a transaction.**
   a. Look for large object or excessive numbers of small object allocations

5. **Configure the right number of MDBs.**
   a. In a Base Server environment, should be less than Thread Settings. The thread settings should be 1 or 2 above the number of MDBs if you want to use other HTTP / IIOP clients.

6. **JVM Heap Size.**
   a. Do GC analysis to gage the optimal heap size for the application.

7. **Application profiling**
   a. In a single-threaded run check how many Objects are being instantiated, how many times methods are called, how many selects / inserts / updates / SP calls being invoked. Understand the transactions nature well and if these numbers at run time don't match the expected values, investigate!!

8. **Pro actively monitor the system and re-assess the settings periodically.**
   a. The application changes over time by functional enhancements, end-user usage pattern differs from the test scripts that were built and tested.

9. **Identify bottlenecks in the infrastructure and application.**
   a. Address them by tuning the system settings and / or re-architecting the application. Follow the flow described in this paper. Find out what systems and products are involved in running your application and how they interact.

10. **Minimize Synchronization in code.**
    a. Difficult to completely avoid synchronization but be very aware of its implications. This point can not be stressed enough. It is very important to know what you are synchronizing and why. The following article talks about synchronization in Java:

11. **Avoid extensive logging and use of println.**
    a. Avoid trace level logging and extensive System.out.println in load test / stress and beyond environments. These are expensive operations and consume a significant amount of resources.

12. **Avoid chatter between distributed components**
    a. Whenever possible all communication should be local, WebSphere on zOS favors the local server and system automatically in many cases. The following paper describes optimal configuration on zOS favoring the local system with benchmark data included:

13. **Reduce Cost of Serialization of deep objects and Co-locate components for high-volume applications.**
    a. Serializing and Deserializing (aka Marshalling / Unmarshalling) of objects before they are sent across the wire is a very expensive operation. Try to minimize the impact of this by declaring fields that can be (re)computed transient. For complex objects customize the serialization by implementing the Externalizable interface.
        a. Information on serializing objects can be found at the following URLs:
        b. With XML serialization is a big issue, the following series of articles will help with performance of XML processing:
14. **Avoid reflection in your Java application where possible. Reflection performs poorly.**
   The following article talks about performance and reflection in Java:
   The following is a review of a book that suggests alternatives to reflection:
   http://www.javaranch.com/bunkhouse/Advanced.jsp
   (Java Reflection in Action by Ira R. Forman and Nate Forman)

**Conclusion**

I would very much like to continue to add to this paper once it is published. Feel free to send more hints and tips or customer examples to meveret@us.ibm.com

Several people took their own time to review and add to this paper and I thank them very much.
Thanks to
   Renuka Chekkala
   Kyle Miller
   Paul Griffiths
   Alan Beaubien