

An IBM WebFacing Tool performance study

Real-world tuning tips and techniques

*Scott Moore, Michael Sandberg
IBM eServer Solution Enablement
December 2005*

Table of contents

Abstract	3
Introduction	3
Test environment	4
About Paragon.....	4
About the IBM WebFacing Tool	4
Hardware	5
Software	5
Test case	5
Images per screen	6
Virtual user script	6
Goals.....	7
Client.....	8
Baseline runs.....	8
Three-tier environment.....	8
Two-tier environment	10
Tuning WebSphere	12
Running with the latest PTFs	12
Verifying the i5/OS settings.....	13
Edge Side Include cache.....	14
Background.....	14
ESI cache in the test case	16
Recommendation.....	17
JVM options.....	18
Initial heap size tuning	18
Java.compiler.....	19
Final runs.....	21
Other optimizations	22
Compression	22
Record definition cache.....	25
Constrained environments	26
Processor-constrained environments.....	26
Running on an underpowered processor.....	26
Partial-processor logical partitions.....	27
Memory-constrained environments	28
Disk-arm-constrained environments.....	30
Start-up and first touch optimizations	30
JSP batch compiler.....	30
JSP pretouch tool	31
Configuring pretouch attributes.....	31
Bytecode cache	33
Using the user classloader cache	34
Summary	35
Resources	36
About the authors	37
Trademarks and special notices	38

Abstract

This paper provides examples of the possible environments used to deploy IBM® WebFacing Tool solutions. The reader will also be walked through techniques to tune the environment to maximize performance.

The IBM WebFacing Tool produces a Web-browser-based user interface for IBM eServer™ iSeries™ applications that use data description specifications (DDS) with RPG and COBOL. The new interfaces are Java™ 2 Platform, Enterprise Edition (J2EE) applications that run in IBM WebSphere® Application Server in addition to the RPG and COBOL applications.

Introduction

This paper is a performance-tuning example that demonstrates the steps readers can take to improve the performance of IBM WebFacing Tool applications running in a WebSphere Application Server on an iSeries system. This paper is not a sizing guide or benchmark.

Through the use of the IBM WebFacing Tool, traditional iSeries applications that use DDS screens (such as RPG and COBOL green-screen applications) can have a graphical, browser-based user interface. This introduces a number of new performance factors to consider, such as the interaction between the IBM HTTP Server and WebSphere Application Server.

Traditionally, RPG and COBOL developers were a distinct group from WebSphere Application Server developers. However, because the IBM WebFacing Tool produces mixed environments of RPG, COBOL, and WebSphere Application Server interactions, a combined set of skills is needed to obtain maximum performance and to avoid performance pitfalls.

This paper will use the Paragon® real-world application simulation to demonstrate the actual steps and results of optimizing performance when combining RPG and IBM WebFacing Tool applications for a number of environments.

Test environment

The following section details the environment used for all tests done for this paper.

About Paragon

In 1983, Metalware was founded to provide metal service centers with a quality computer system and software solutions. Since then, the business has been dedicated to providing the most comprehensive information systems and production technology solutions expressly for metal service centers nationwide.

The firm has grown because of its innovative approaches and dedicated professional staff. In 1988, Metalware became an IBM Premier Business Partner. Coupled with midrange systems from IBM, Metalware provides its customers with the excellent solutions in a multiuser environment. The workhorse of Metalware systems is the IBM iSeries family of servers (formerly the IBM AS/400® family of systems), recognized with the Malcolm Baldrige National Quality Award in 1990.

Metalware has completed more than 280 installations and has invested more than \$4.5 million in research and development.

About the IBM WebFacing Tool

The IBM WebFacing Tool helps to convert DDS display file source members quickly so that the user interface for iSeries programs can run in a browser. The IBM WebFacing Tool facilitates the conversion process through user-friendly wizards for selecting the DDS source for the 5250 application, converting it, and deploying the new Web-based interface to the program as a WebSphere application.

JavaServer Pages™ (JSPs™) and extensible markup language (XML) files are generated at development time and take the place of the DDS code, allowing for flexibility in customizing the appearance of a new interface prior to run time. Using the IBM WebFacing Tool style properties, attributes in the pages can be changed, such as graphics, fonts, colors, and layouts. To change a previously converted application, simply reconvert it, and select a new style.

WebSphere Development Studio Client Advanced Edition for iSeries Version 6.0 offers the following capabilities with the IBM WebFacing Tool:

- **System screen support:** System screens previously had to be Web-enabled using a 5250 data stream conversion tool and required an interactive workload. System screen support with the IBM WebFacing Tool enables developers to Web-enable entire applications, including user interface manager (UIM)-based iSeries system screens (and other UIM screens), which can then be run entirely in batch mode using the IBM OS/400® commands.

- **Portal support:** The IBM WebFacing Tool now allows the developer to generate Struts-based portlet applications and to add a portlet created by the IBM WebFacing Tool to an existing non-Struts-based portlet project. These applications can then be deployed to a WebSphere Portal server and added to the portlet pages served by that server. (**Note:** Struts is an open-source framework for building Java Web applications.)
- **Single sign-on:** Single sign-on enables users to access more than one application and multiple platforms with one user ID and password. For example, applications that have been enhanced with the IBM WebFacing Tool and other Web tools can be integrated. Then, a user can securely access the applications without requiring separate IDs and passwords for each.

Hardware

The hardware used for the tests consisted of two systems, outlined in Table 1. For each section of this paper, these tests are identified using the machine names shown in this table. Notice that these machines both have an extraordinary amount of memory and disk arms. The section of this paper labeled [Constrained environments](#), examines situations where the amount of CPU, memory, and disk arms are reduced. The resources in these machines are excessive. For the purposes of Web-faced application performance, these numbers can be reduced.

Machine name	Model / processor feature	Processor family	CPW rating	Memory	Disk arms
i5 570 2-way	i570	POWER5™	6000	15 GB	90
i5 270 2-way	i270 / 2434	POWER3™	2350	15 GB	12

Table 1: Machines involved with test case

Software

All of these tests were running a Web-faced application built with the WebSphere Development Studio Client Advanced Edition for iSeries V6.0.0. The WebSphere version was WebSphere Application Server V6.0.2 base edition. The same behavior and results are expected running the Express Edition.

The examples in this paper used WebSphere Application Server V6 to demonstrate performance tuning steps, but none of the tuning was specific to WebSphere Application Server V6.

Test case

The application used to drive the workload came from Paragon Consulting Services. The application is the core product for the Paragon solution with some images added

to the screens to make the test environment more realistic to an actual, deployed solution.

Some definitions that will be used to describe the workload are:

- **Test case:** This is the environment that simulated load on the Web-facing application.
- **Screen:** This is a single Web page that a user will see in the browser. For example, a screen can be a single URL request, such as: <http://www.ibm.com>, or can be a Web page that is the result of following a link on a previous Web page.

Images per screen

Each screen that was served displayed a random number of images in one of three groups: small, medium, and large images. In Table 2, the test case averaged six images per screen for a total average of 75.5 kilobytes in size per screen.

Image Size	Number of unique images on site	Min displayed / page	Max displayed / page	Average size / page
5 KB	400	2	7	22.5 KB
25 KB	70	0	2	25 KB
56 KB	30	0	1	28 KB
Totals	500	2	10	75.5 KB

Table 2: Images per screen in test case

Virtual user script

Figure 1 shows a high-level view of the script that was run for these measurements. The path is shown that each virtual user took as it executed the script. Each user paused three to nine seconds (randomly) between screens to simulate a user hesitating between screens. The number of users for each test scenario are documented in each test scenario description.

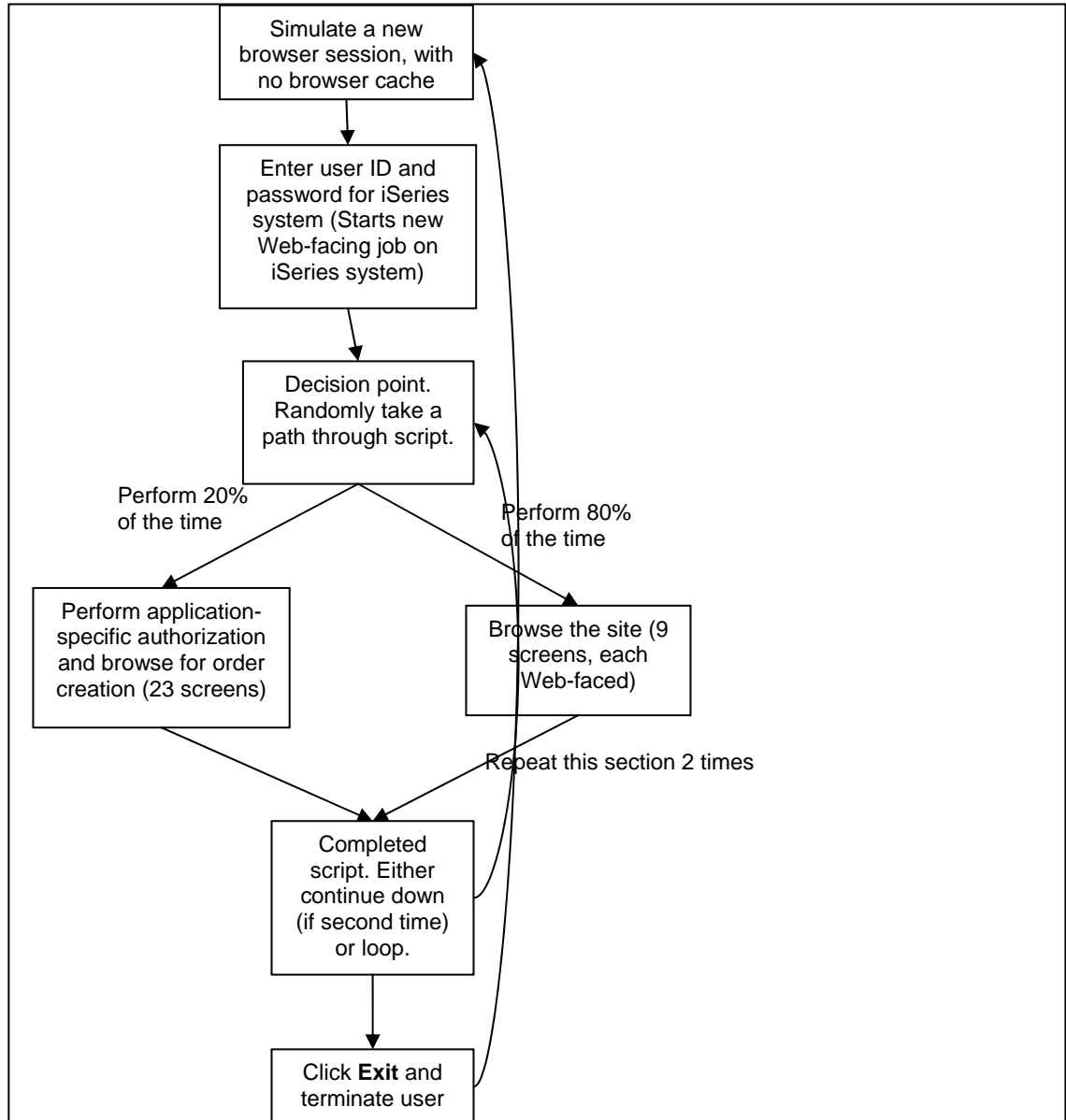


Figure 1: Virtual user script

Goals

The first goal was to sustain 500 concurrent users with an average one-second response time in two-tier tests running on a 2-way IBM POWER5™ machine.

The second goal was to sustain 500 concurrent users with average one-second response time in three-tier tests running WebSphere Application Server and the IBM HTTP Server on a 2-way IBM POWER3™ machine. In this case, the back-end application was run on the same 2-way POWER5 machine above.

Client

The PC client had a bigger impact than expected. Many people thought that, because the PC client was only running a browser, an old Intel® Pentium® II processor was sufficient. However, a slow client PC can degrade response times significantly. Testing a Web-faced application showed an increase in response time of a half-second per page on a slow client PC. This means a 50% improvement in performance might be realized using the same application and network, but a faster client PC. A client PC with a minimum of a Pentium (or AMD® Athlon) 1.0-gigahertz processor and 512 megabytes of RAM eliminates most performance issues on the client.

To state the obvious, the fastest data transmitted is the data that is never transmitted. This can be accomplished by ensuring that the client's browser settings are caching data properly. Because HTML pages are almost always a series of requests for text and images, much of this data can be cached at the browser instead of being downloaded every time a page is returned to the user. When testing, make sure that browser caching is turned on. This is done on the Microsoft® Internet Explorer browser by selecting the configuration button named:

**Check for newer versions of stored pages:
Automatically within Internet Explorer Temporary Internet files**

To highlight performance enhancements in a particular section of the test, this environment was altered from these defaults. When done, details of the deviations from the test case are described. The test scenario results do not include the time it took the PC to render the pages.

Baseline runs

The baselines were performed with the Paragon application deployed on an iSeries system with all the standard defaults. The instance was created from the defaultnosamp template shipped with WebSphere V6.0.

Three-tier environment

The three-tier environment involves three machines: a user machine runs the browser, another machine runs the HTTP server and WebSphere Application Server, and a third machine runs the iSeries jobs. Figure 2 illustrates this environment.

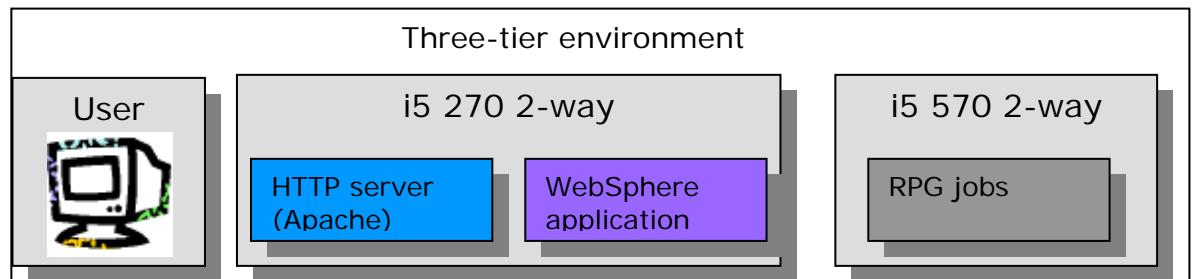


Figure 2: Three-tier environment

Initially, 500 concurrent users were run against the environment to determine initial results. From the results of the benchmark depicted in Figure 3, it became apparent that the default environment cannot sustain 500 users at the same time.

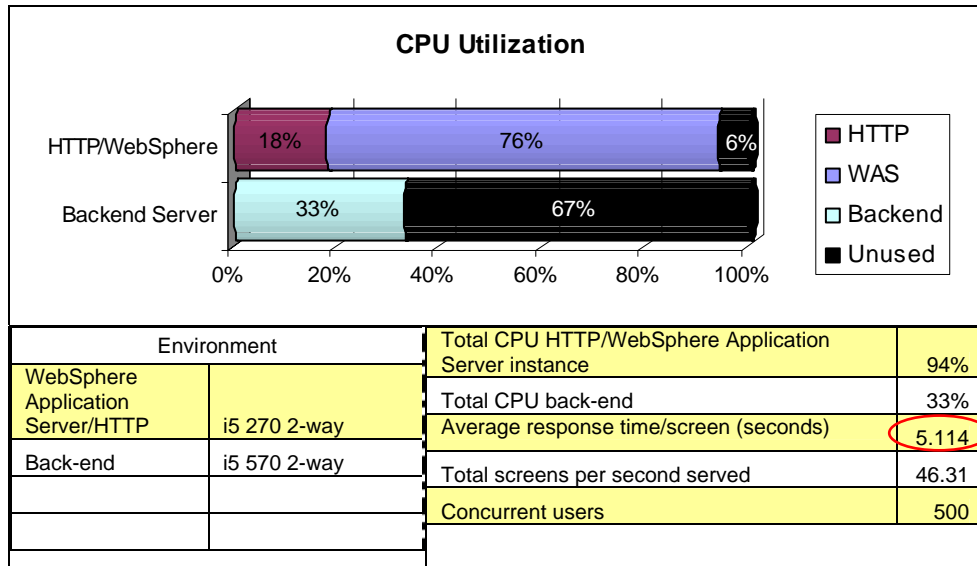


Figure 3: CPU utilization for 500 users in three-tier test

The results for this run are shown in the figure in three sections. The top section is a view of the environment’s CPU usage. In the test, two machines were involved, with the third tier being the client machine running the browser. In the baseline (Figure 3), 18% of the CPU was consumed in the HTTP server job. 76% of the CPU was spent in the WebSphere job, and 6% of the CPU was idle or unused. On the back-end machine, 33% of the CPU was spent in the RPG jobs. The bottom-left section describes the environment. In this case, there were two machines involved: a 2-way i5 270 and a 2-way i5 570.

The bottom-right part of the figure contains a few key metrics stating the health of the run. The **Total CPU HTTP/WebSphere Application Server instance** and **Total CPU back-end** show the CPU usage on these two machines. The **Average response time/screen (seconds)** is the average number of seconds it took the server to serve the page back to the client machine. It does not include the time it took for the client to render the page. The **Total screens per second served** gives an average of how many requests were served each second. **Concurrent users** is the total number of simulated clients running against the machine at the same time.

Because of unacceptable response times of 5.114 seconds when running the test with 500 users, the decision was made to perform the first runs with only half of the required number of users. Figure 4 demonstrates the baseline results running in the three-tier environment with 250 users.

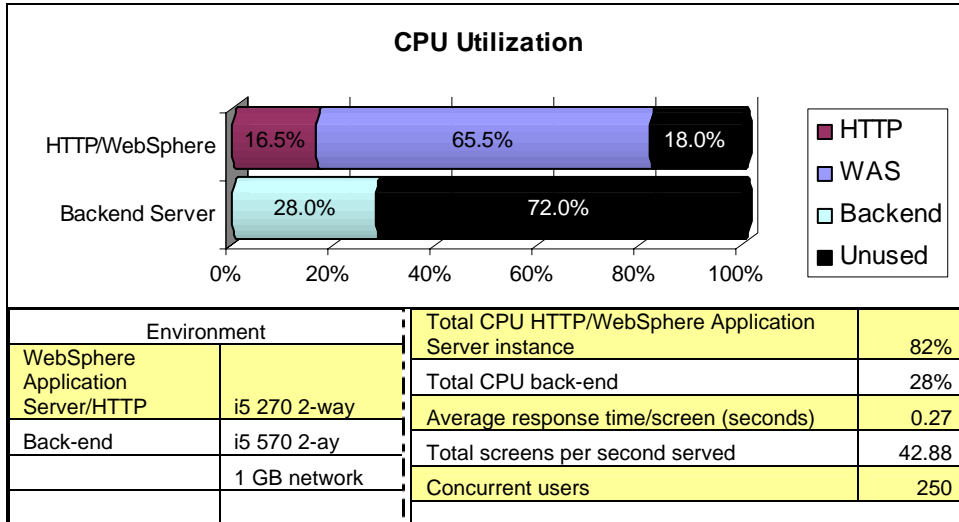


Figure 4: CPU utilization for 250 users in three-tier test

This environment behaved much better than the 500-user run, with response times averaging 0.27 seconds. Therefore, this is the starting point that will be used to run further tests.

Two-tier environment

The two-tier environment consists of two machines. One is the user machine that is running the browser. The second is the IBM HTTP Server and WebSphere Application Server, and it runs the back-end jobs. Figure 5 shows this environment, which is what most users of the IBM WebFacing Tool run. Because it is easier to detect improvements in the WebSphere jobs in the three-tier test, the focus is on the changes in the three-tier environment. Later in this paper, the two-tier test is repeated with the same changes that were made for the three-tier environment.

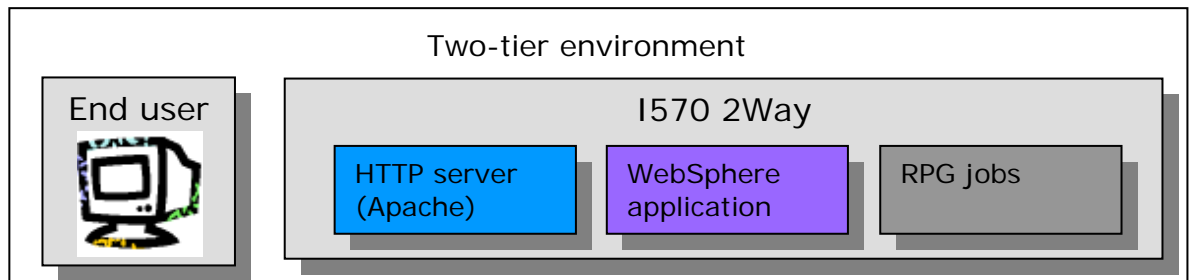


Figure 5: Two-tier environment

A total of 500 concurrent users were run against the two-tier environment to determine initial results. From the results of this benchmark, as shown in Figure 6, it was apparent that the default environment cannot sustain 500 users at the same time.

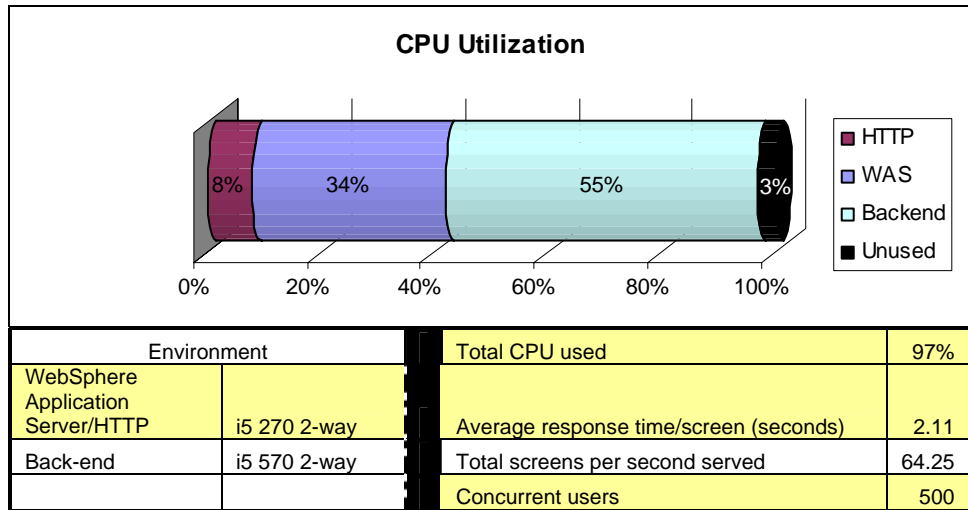


Figure 6: CPU utilization for 500 users in a two-tier environment

Note that the CPU is maxed out at 97% utilized, and the average response time fails the one-second requirement. Just like the three-tier test, the two-tier test was backed down to 250 users and run again. The results, as depicted in Figure 7, fell within the acceptable range.

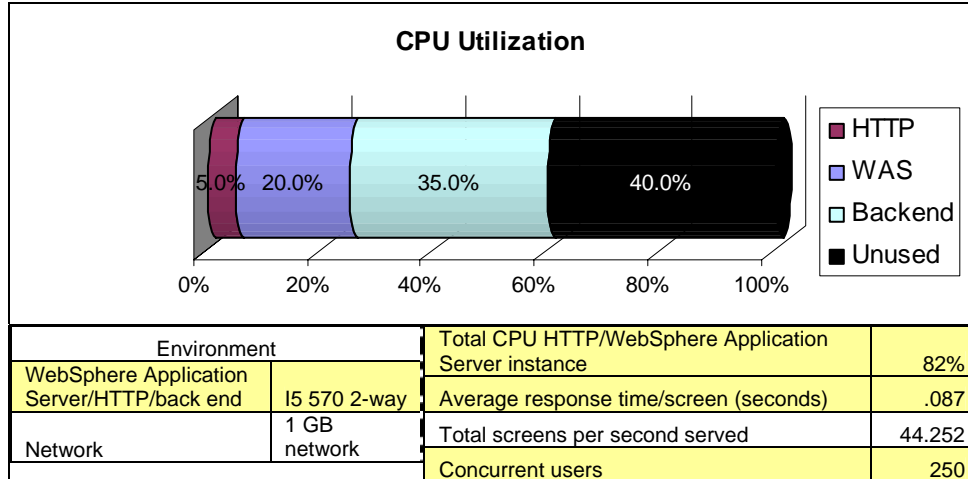


Figure 7: CPU utilization for 250 users in a two-tier environment

At first glance, the numbers for the two-tier tests look better than those in the three-tier tests. This is primarily because the two-tier tests used the POWER5 processor for all of the transaction, whereas the HTTP/WAS ran on a POWER3 processor in the three-tier environment.

Tuning WebSphere

In order to improve all-around efficiency, several measures can be taken to fine tune WebSphere Application Server.

Running with the latest PTFs

Table 3 lists the relevant program temporary fix (PTF) information available for running the IBM WebFACING Tool on the iSeries system.

Software	Available via
WebSphere Development Studio Client Advanced Edition for iSeries	WebSphere Development Studio Client Advanced Edition for iSeries can be updated in two ways: <ul style="list-style-type: none"> • Use the IBM Rational® Product Updater (RPU) that is part of the WebSphere Development Studio Client Advanced Edition for iSeries install. • Access the Support section of the WebSphere Development Studio Client Advanced Edition for iSeries Web site.
IBM WebFACING Tool runtime (STRTCPSVR *WEBFACING)	IBM WebFACING Tool iSeries Server PTFs www.ibm.com/support/docview.wss?rs=0&uid=swg27002213
Latest i5/OS or OS/400 group HIPER PTFs	If IBM support is available, send a request for the latest i5/OS or OS/400 group HIPER PTFs via the IBM fixes Web site listed in the Resources section of this paper. If current support is not through IBM, contact the appropriate IBM Business Partner. <ul style="list-style-type: none"> • OS/400 V5R2: SF99519: 520 Group HIPER • i5/OS V5R3: SF99529: 530 Group HIPER
Latest CUM package	CUM packages can be requested via the IBM fixes Web site listed in the Resources section of this paper.
Latest Java Group PTF	Java support is included with i5/OS and OS/400. To access the latest Java group PTFs, follow the process described above. <ul style="list-style-type: none"> OS/400 V5R2: SF99169: 520 Java i5/OS V5R3: SF99269: 530 Java
Latest IBM DB2® UDB group PTFs	IBM DB2 Universal Database™ (DB2 UDB) for iSeries is included with i5/OS and OS/400. To access the latest DB2 UDB for iSeries group PTFs, follow the process described above. <ul style="list-style-type: none"> • OS/400 V5R2: SF99502: 520 DB2 UDB for iSeries • i5/OS V5R3: SF99503: 530 DB2 UDB for iSeries
WebSphere Application Server Express Version 5.1.2 or greater (V6 releases are preferred)	Included with all shipments of i5/OS V5R3 or greater: <ul style="list-style-type: none"> • 5722-WE1: WebSphere Application Server Express 5.0 and 5.1 • 5722-WE2: WebSphere Application Server Express 5.1 and 6.0
WebSphere Application Server Group PTFs	For details on WebSphere group PTFs and details on supported i5/OS releases, see the Resources section of this paper.

Table 3: PTF information

Verifying the i5/OS settings

This section will highlight the settings that effect performance on the iSeries platform that were used throughout the test. Table 4 lists these system values and settings.

System Value	Description	Recommended setting
QPRCMLTTSK	Allows more than one set of task data on each CPU	1 or 2 on POWER5 machines
QTHDRSCADJ (i5/OS V5R3 and later)	Determines if the system will make adjustments to the affinity of threads	1
QTHDRSCAFN (i5/OS V5R3 and later)	Specifies whether secondary threads have affinity to the same group of processors	Group: *NoGroup Level: *Normal
QMAXACTLVL	Total number of threads that can compete at same time for memory and processor	*NOMAX or large value
QPFRAJ	Specifies whether the system automatically adjusts values for max active and pool sizes	0 (If not 0, avoid falling below the minimum memory needed in the pool)

Table 4: Performance settings

For machines running many workloads, isolating the WebSphere jobs (running in QWAS6 subsystem) in a separate shared pool is ideal. For the subsystem and pool that are running the Java Virtual Machine (JVM™), use the system command WRKSYSSTS to view and change settings shown in Table 5.

Pool setting	Description	Recommended
Max Active	Maximum number of threads that can use the processor concurrently	This value must be large enough to avoid any threads entering ineligible state. The recommendation is to set max active above 200 for the pool running WebSphere.
Pool size	Amount of main store allocated for threads running in the pool	This value must be large enough for all JVMs running in the pool and at least 768 megabytes for each WebSphere instance running concurrently. For ideal performance, allocate at least 1 gigabyte of memory for each instance.

Table 5: WRKSYSSTS settings

Figure 8 shows a screen shot of the Work with System Status display.

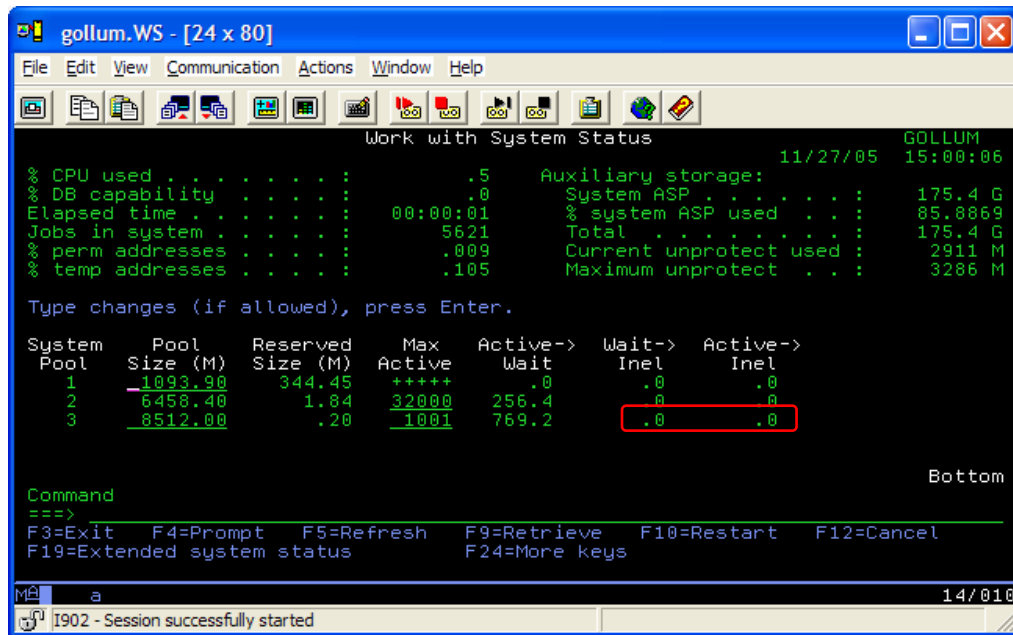


Figure 8: Work with System Status display

Use the WRKSYSSTS command and hit **F11** once to determine if threads are entering ineligible state. The user must be either at the intermediate or advanced assistance level to display the ineligible columns.

Edge Side Include cache

The first enhancement discussed here uses the Edge Side Include (ESI) cache in the WebSphere plug-in for the IBM HTTP Server.

Background

An application containing a variety of J2EE modules is packaged as a single file known as an Enterprise Archive (EAR) file. Included in the J2EE module is a Web Archive (WAR) file that contains Web-related components, including static elements, such as HTML files and graphics, as well as servlets, Java Server Pages, and other Java classes. In most Web-facing environments, the EAR file is deployed to an application server, and the entire application is packaged and served from the file. This includes the static content, which is handled by a WebSphere Application Server facility called the file serving enabler.

Consider a Web request sent from a browser for a simple HTML page with a few embedded graphics from a Web-faced application with WebSphere Application Server file serving enabled. The HTTP server will see the request, find the pass rule in the plug-in file, and correspondingly forward the request to WebSphere Application Server. At the Web container, the file serving enabler will read the file from the

directory where the application has been deployed and send the request back. The user's browser will parse the HTML file as it receives it and will then display it. During this process, the browser sees the graphics files needed to render the page. It sends requests for the files, and they will be handled similarly by the file serving enabler.

Although this is the simplest deployment scenario, it did not produce the most efficient performance until the ESI cache became available in WebSphere V5.0. The plug-in can use ESI to cache static content at the Web server where the plug-in is installed, essentially acting as if it were served from the Web server's static cache. However, the benefits of using the plug-in are that static content can remain in the WAR, simplifying packaging. Caching at the plug-in reduces the burden on the file-serving facility at the application server, as shown in Figure 9. This method does not perform as well as splitting the static content off to the Web server, but it is less administrative work for developers who need to avoid the additional deployment steps.

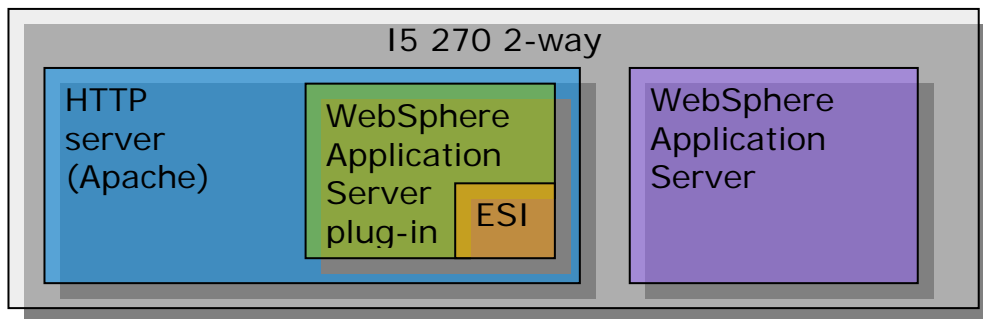


Figure 9: ESI cache

The plugin-cfg.xml file is contained in...

```
{WAS Application Root}/config/cells/{cell Name}/nodes\IHS_{Node Name}_node\servers\IHS_{Node Name}
```

Starting in WebSphere V5.1, the ESI cache is enabled. The size is set to 1024 kilobytes (1 megabyte) of memory in the HTTP server. When this space is filled, entries will be purged from the cache based on their pending expiration. Entries closest to expiration are purged first. The **invalidation monitor** setting monitors messages from WebSphere Application Server that indicate when a given entry or group of entries become invalid and need to be purged from the cache. It also monitors entries in static cache and accumulates other statistics. The default entries in the plugin-cfg.xml file for configuring the ESI cache appear in the following code:

```
<Property Name="ESIEnable" Value="true"/>
<Property Name="ESIMaxCacheSize" Value="1024"/>
<Property Name="ESIInvalidationMonitor" Value="false"/>
```

Static entries in the ESI cache time out every 300 seconds by default. This can be changed by entering the following command in the application server's JVM command line parameters:

```
-Dcom.ibm.servlet.file.esi.timeOut=120
```

In this example, the default was changed from 300 seconds to 120 seconds. This change is also reflected in the generic JVM arguments in the admin server As shown in Figure 10. These tests were run with the default setting on the cache timeout.

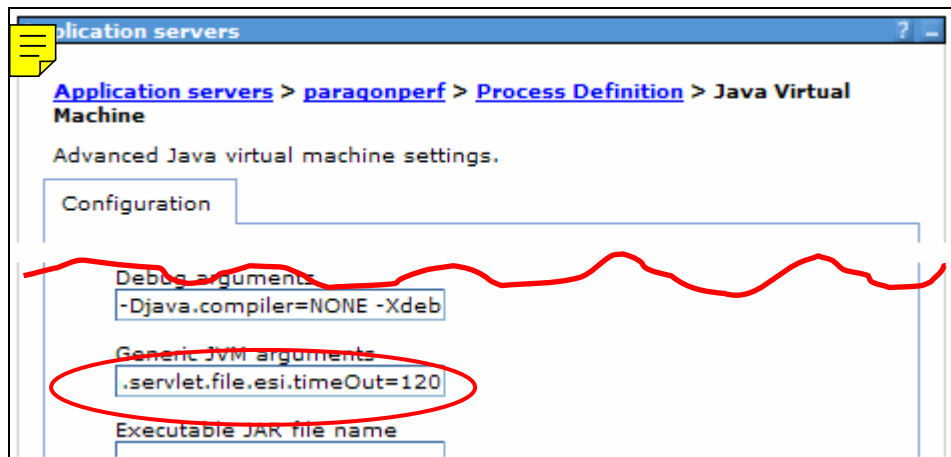


Figure 10: Time out change in the generic JVM arguments

ESI cache in the test case

Because WebSphere 6.0 is running, the ESI cache is enabled by default when associating the IBM HTTP Server to the WebSphere instance. With WebSphere V5.0, the ESI cache has to be configured by hand by modifying the plugin-cfg.xml file. In order to take advantage of the ESI cache, requests must be made through the HTTP server, instead of going to the internal HTTP port in WebSphere.

Because the defaults automatically enable the ESI cache, the baseline tests performed for this demonstration had the ESI cache enabled. By default, the ESI cache is only 1 megabyte, and 500 static images are being served for a total of 5478 kilobytes of images. Thus, the default of 1 megabyte was not enough, and the cache size needed to be set to a larger number than the default. By changing the size to 8 megabytes, the cache was able to serve the images during the run of the test. The following command was issued to change the size:

```
<Property Name="ESIMaxCacheSize" Value="8192" />
```

Figure 11 below displays the details of this test:

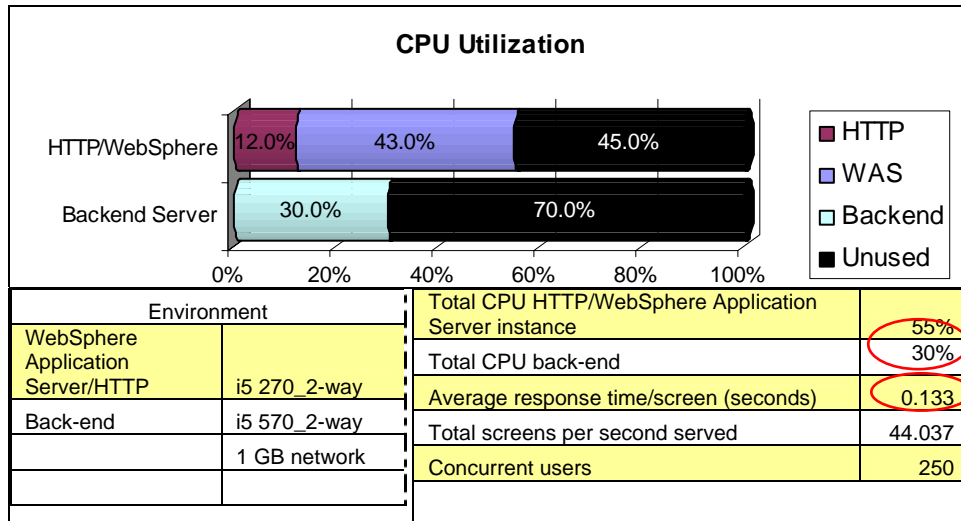


Figure 11: ESI cache of 8 megabytes

The important numbers are circled in red. The total CPU usage for the HTTP and WebSphere machine dropped dramatically, going from 82% CPU consumption to 55% for an improvement of 33%. Average response time dropped from 0.270 to 0.133 seconds (an improvement of 50%). Both improvements are attributed to having a large enough ESI cache to hold all of the static images used by the application server.

Recommendation

Utilize the ESI cache from the application, especially if static files (images, documents, and other files types) are served from WebSphere. Ensure that the cache is large enough to hold the majority of these objects.

JVM options

Several JVM options can improve performance.

Initial heap size tuning

Two parameters influence garbage collect (GC) on the iSeries platform: initial heap size and maximum heap size. In WebSphere V5 and beyond, the initial and maximum heap sizes are set via the WebSphere administrative console.(see Figure 12.)

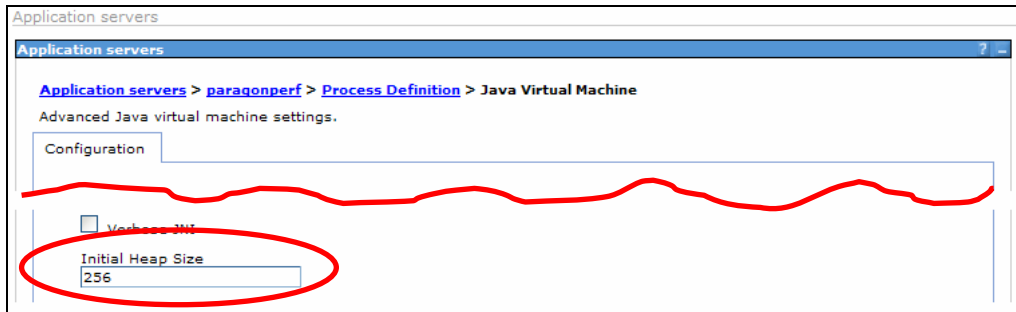


Figure 12: Initial heap size in the administrative console

Even though other platforms also have these parameters, the behavior on the iSeries platform is unique. The initial heap size can be more accurately called the GC threshold, as it determines how often the garbage collector thread will run. For example, if a process is running with an initial heap size of 100 megabytes, the GC will cycle every time 100 megabytes worth of objects have been created. Therefore, increasing this value makes the GC run less frequently, allowing the heap to grow larger between collections. In turn, each GC cycle is lengthened because it must scan more memory.

The maximum heap size limits the Java heap to a specific size. When an application requests an object, and the JVM has to allocate more memory in the Java heap to contain the object, the JVM ensures that the allocation will not exceed the maximum. If the JVM cannot allocate the additional memory, the GC will run synchronously, and it will try to collect unused space within the heap. If the object cannot be allocated after the synchronous GC, a `java.lang.OutOfMemory` exception will occur.

The appropriate value for the initial heap size is dependent on the environment and application. Tuning the GC requires balancing the frequency and duration of cycles. The paper **Tuning garbage collection for Java and WebSphere** provides more information on tuning the GC.

Because this environment is not constrained by memory, initial heap size was set to 256 megabytes, and Max Heap size was set to NOMAX. Figure 13 shows the results of this change:

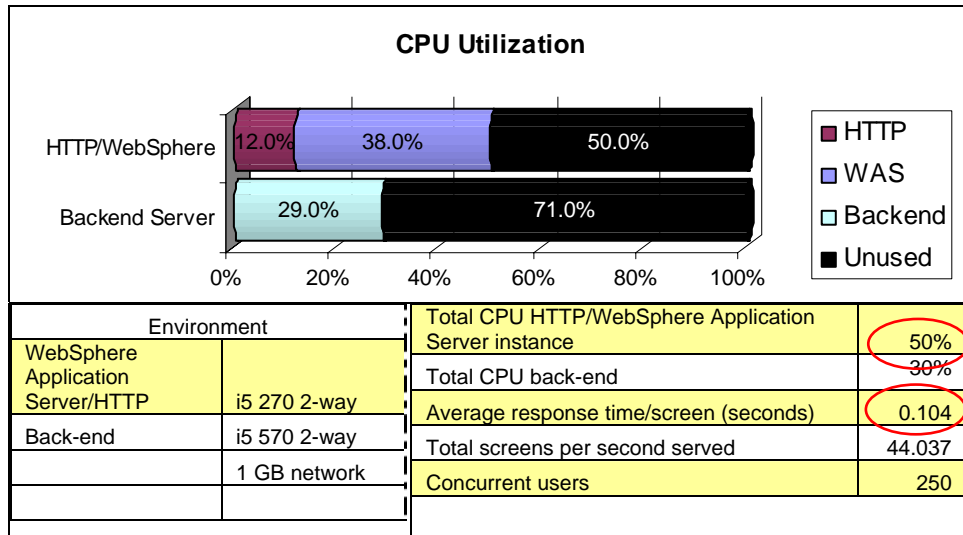


Figure 13: Initial heap size is 256 megabytes

The red circles in Figure 13 indicate the important numbers. The total CPU usage for the HTTP and WebSphere machine dropped noticeably, going from 55% CPU consumption to 50% for an improvement of 9%. The average response time dropped from 0.133 to 0.104 seconds for an improvement of 22%.

Java.compiler

The iSeries JVM has two main modes on running Java classes, direct execution, and Just-In-Time mode. By default, the JVM will run a class in direct execution mode if a direct execution program exists. However, if none exists, the JVM will run that class in JIT mode. Starting in OS/400 V5R2, the JIT compiler produces more efficient code than direct execution programs. In i5/OS V5R3 and future releases, the performance improvements are even more apparent. Thus, you can force JIT compiles on all classes, even if direct execution programs exist. You can accomplish this by adding `-Djava.compiler=jitc` to the generic JVM arguments, as shown in Figure 14:

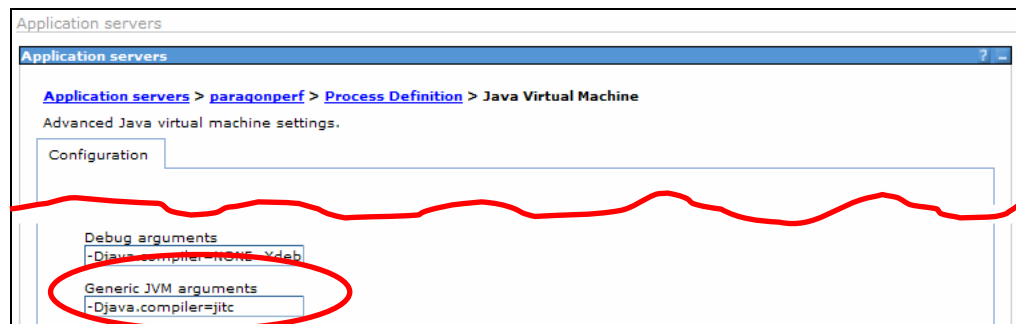


Figure 14: Force JIT compiles

After this change is made, the improvements shown in Figure 15 were discovered:

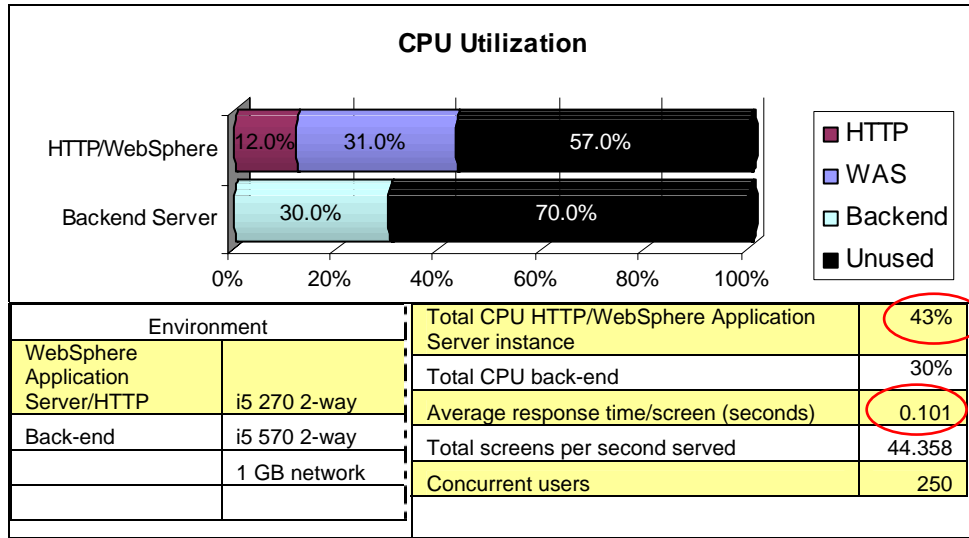


Figure 15: JIT mode

The improvements in both CPU usage and response time can be attributed to forcing all Java classes to run in JIT mode. In i5/OS V5R3, the JIT compiler produces more efficient code than direct execution, as these results demonstrate. The important numbers are again circled in red. The total CPU usage for the HTTP/WebSphere machine dropped noticeably, going 50% CPU consumption down to 43% for an improvement of 14%. The average response time dropped from 0.104 seconds to 0.101 for an improvement of 3%.

Final runs

After making the changes documented above, a run with 500 users was attempted in both a three-tier environment and a two-tier environment. You can see the results of the three-tier environment in Figure 16:

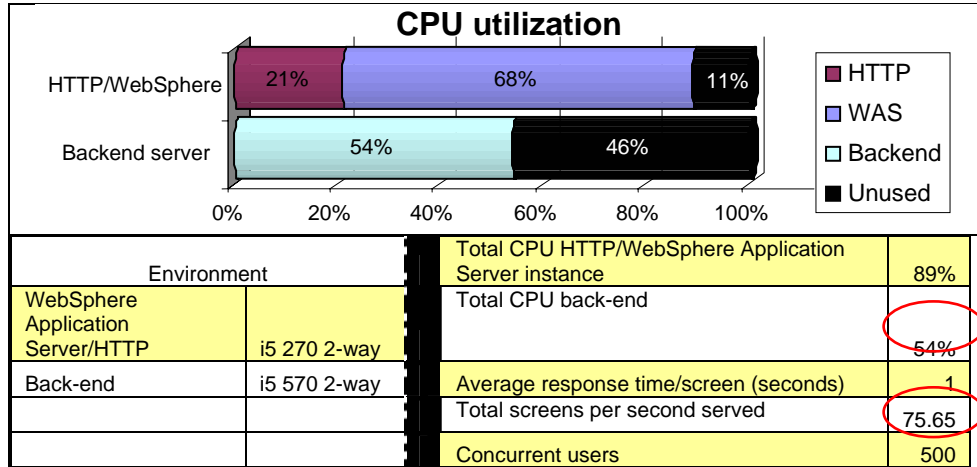


Figure 16: 500 users in a three-tier environment using JIT mode

Two important metrics were examined. The first was whether one-second response times had been achieved with 500 users. Not only was this a success, but compared to the initial baselines, average response time per screen decreased 80%. The second metric indicated whether the screens served per second had increased sufficiently. As compared to the original measurement, there was a 63% improvement.

In the two-tier test, the response time test also improved. Figure 17 shows the results.

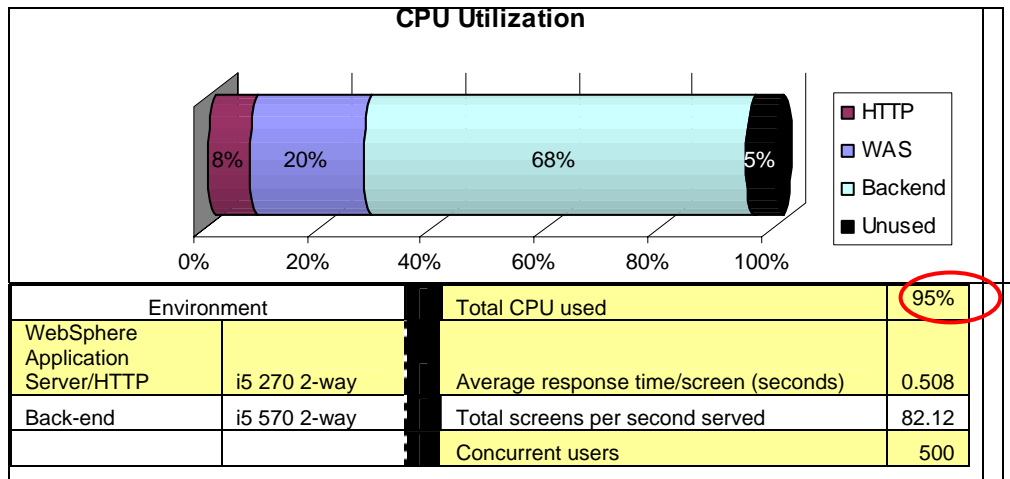


Figure 17: A two-tier environment using JIT mode with 500 users

Compared to the initial baselines, average response time was reduced 75% per screen, and capacity increased by 27%.

Other optimizations

Other options exist to optimize the system. This section will discuss those methods.

Compression

LAN connection speeds and Internet hops can have a large impact on page response times, even when a fast server is being used.

It is very common for a browser page to contain between 15 and 75 kilobytes of data. Users running a Web-faced application over a 256-kilobyte Internet connection might find results unacceptable. If every screen averages 60 kilobytes, the time for that data spent on the wire is significant. If several users are using the application simultaneously, then the page response times will be even longer.

Two options are available to support HTTP compression for Web-faced applications, and both will significantly improve response times over a slow internet connection. Compression can be set in the IBM HTTP Server (Powered by Apache), or by using a compression filter shipped with the Web-facing application. By default, starting in Version 5.0 of the IBM WebFacing Tool, the compression filter is activated, and all HTML returned by WebSphere will be compressed.

This compression does waste CPU to get the pages in condensed format. In some cases, it is wise to turn off compression (for example, if network bandwidth is not a concern, or if most of the users are on high-speed connections). This is especially true for intranet implementations where all users are on the LAN.

To turn off compression, comment out a few lines in the deployment descriptor web.xml. For applications built in WebSphere Development Studio Client, web.xml can be found in the **WF Projec > WebContent > WEBINF** directory (Figure 18).

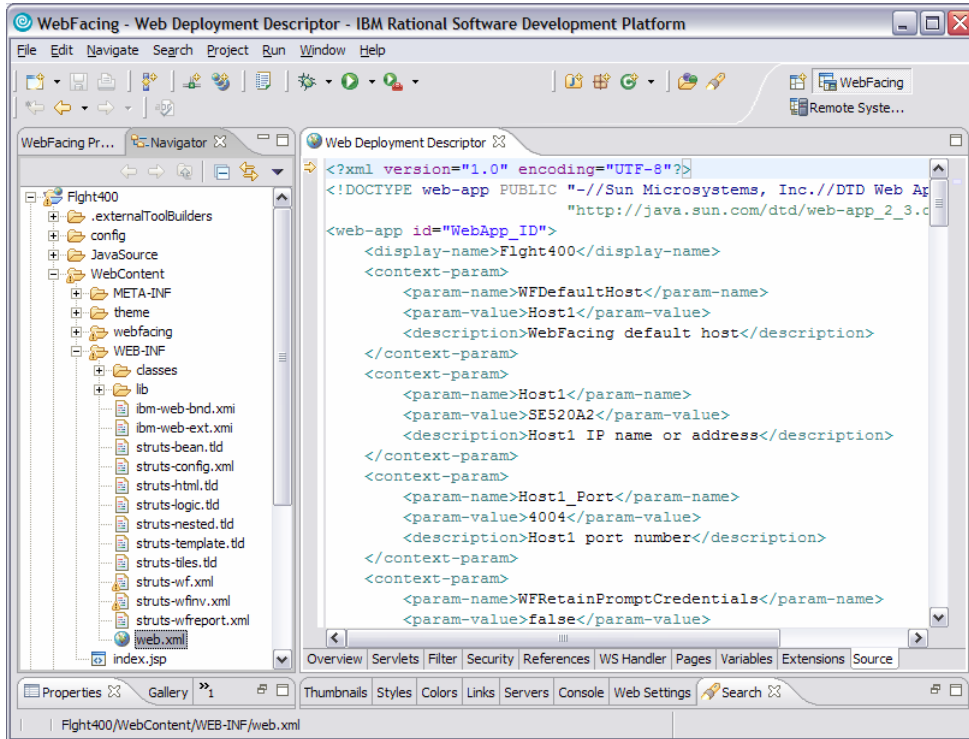


Figure 18: Finding web.xml

The default path for web.xml that is modified in deployed applications will be located in the following directory for WebSphere Application Server V6.0 applications:

```
/QIBM/UserData/WebSphere/AppServer/V6/<edition>/profiles/<Profile
Name>/config/cells/<cell Name>/applications/<Application
Name>.ear/deployments/<Application Name>/<Application Name>.war/WEB-
INF directory
```

First, locate the filter (with filter-name CompressionFilter) and filter-mapping (with filter-name CompressionFilter), and comment these out with the `<!-- text -->` XML comments. For example:

```
<!-- filter>
  <filter-name>CompressionFilter</filter-name>
  <display-name>CompressionFilter</display-name>
  <description>WebFacing Compression Filter</description>
  <filter-class>
    com.ibm.etools.iseries.webfacing.runtime.filters.CompressionFilter
  </filter-class>
</filter -->
```

```
<!-- filter-mapping>
  <filter-name>CompressionFilter</filter-name>
  <url-pattern>/webfacing/WebFacing.do</url-pattern>
</filter-mapping -->
```

Because the network is not constrained, the 500-user run was attempted again with compression turned off. Figure 19 shows the results:

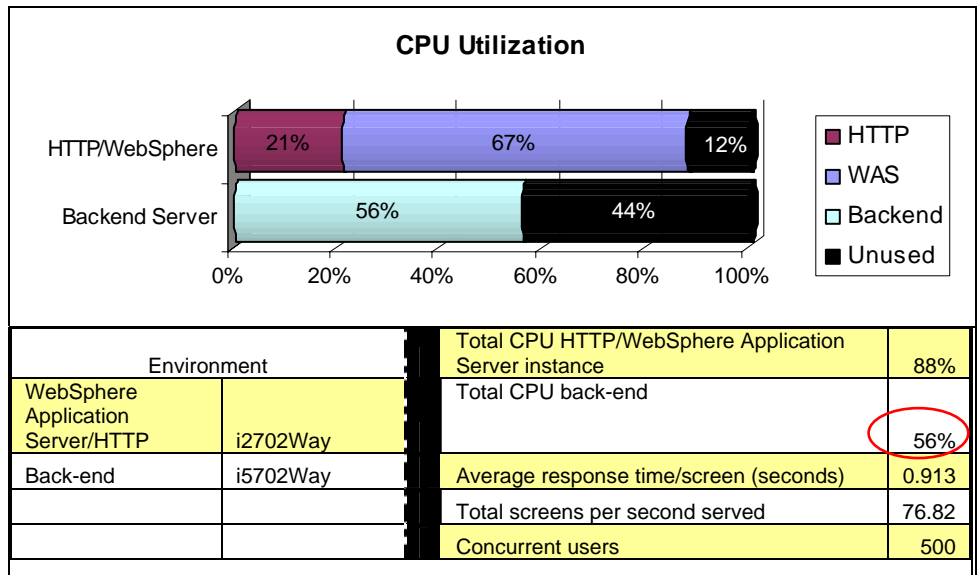


Figure 19: 500 users with compression turned off

The most important change in this run was that the response time decreased by approximately 9%, and there was a slight drop in the CPU usage on the HTTP/WebSphere Application Server instance.

Record definition cache

When set to an appropriate level for the Web-faced application, the record definition cache can significantly decrease memory usage and slightly decrease processor usage. The number of record definitions that the cache will retain is set by an initialization parameter in web.xml, the application's deployment descriptor. By changing the cache size, the application can be tuned for optimum performance and minimum memory requirements. The effects of the cache size on the application are summarized in Table 6.

Cache size	Effect
Too small	<ul style="list-style-type: none"> ▪ Performance effected adversely ▪ Definitions discarded before being re-used ▪ Significant overhead to create record definitions
Correct	<ul style="list-style-type: none"> ▪ 90% of accessed record data definitions retained ▪ Few cache misses for records not commonly used
Too large	<ul style="list-style-type: none"> ▪ All record data definitions cached ▪ Memory wasted on seldom used definitions

Table 6: Effects of cache size on an application

In order to determine the correct size for a given application, the number of commonly used record formats needs to be estimated. The default size is 600. To set the cache size to something other than the default, add a session context parameter in the web.xml file. In the following example, the cache size is set to 200 elements, which might be appropriate for a very small application.

```
<context-param>
<param-name>WFBeanCacheSize</param-name>
<param-value>200</param-value>
<description>WebFacing Record Definition Bean Cache Size</description>
</context-param>
```

Note: It is also possible to edit the web.xml file of a deployed application. Typically, this file will be located in the following directory for WebSphere Application Server V6.0 applications:

```
/QIBM/UserData/WebSphere/AppServer/V6/<edition>/profiles/<Profile
Name>/config/cells/<cell Name>/applications/<Application
Name>.ear/deployments/<Application Name>/<Application Name>.war/WEB-
INF directory
```

In previous versions of the IBM WebFACING Tool, two servlets were enabled to assist with managing the record definition cache. One displayed the elements currently in the cache, and the other loaded the cache. To prevent misuse or exposure of data, neither servlet was traditionally enabled in a Web-facing application. These two servlets are not currently supported in WebSphere Development Studio Client for iSeries V6.0.0.

Constrained environments

The tests conducted were on machines with large amounts of CPU, memory, and disk arms. The reality of most environments is that the workload must run on a machine that has either a constrained environment or other tasks running.

Processor-constrained environments

In general, there are two main processor-constrained environments. The first is when a processor is either underpowered or outdated. The second is when other partitions running on the system are using portions of the processor. Thus, running in a processor-constrained environment can have the largest impact on perceived performance.

Running on an underpowered processor

Typically, when developers think about the power of an iSeries server, they look at the commercial processing workload (CPW) rating, which is a measure to compare iSeries server models in terms of how efficiently a typical workload can be processed.

CPW is a good way to determine the capacity of the machine, but it is not a good indicator of the user response times in a Java or WebSphere Application Server environment. For example, a 1-way POWER5 iSeries system with a 3300 CPW rating, and a 12-way IBM PowerPC® AS system with a 4550 CPW rating will provide radically different response times. Theoretically, the capacity between these machines will be similar, but much better response times can be expected from the POWER5 machine.

A thread running in a JVM can only perform on one processor at a time, and a single-user request will perform its processing in one thread. Therefore, a single-user request will only perform on one processor at a time, which means the user is dependent on the speed of a single processor to determine how quick the response will be.

To provide an estimate of what a user will experience, divide the CPW rating of the machine by the number of processors. The 1-way iSeries POWER5 system mentioned above has 3300 CPW per processor. In comparison, the 12-way iSeries PowerPC AS system only has 380 CPW per processor. Therefore, the POWER5 system has roughly eight times the CPU power to process a single Java or WebSphere user or thread. A response time that is up to eight times faster can be expected.

However, this is only a rough estimate, and there are a few problems with the formula. For instance, POWER5 hardware offers simultaneous multithreading (SMT) mode, which allows two threads of execution to run on one processor at the same time, with one as the primary thread and the other as the secondary thread. If the processor is

fully utilized, two threads need to be actively using the processor at the same time. If both threads are actively consuming the CPU, the primary thread will use 80%, and the secondary thread will use the remaining 20%. For more information about SMT mode, see the **Resources** section of this paper.

Another problem is that a JVM always has other threads of execution, which can run on other processors if available. For instance, the job can have GC threads that run with varying frequency depending on the application.

In terms of performance, a 1-way 3300 CPW system is the standard by which other iSeries processors are compared. If the system has less than 3300 CPW per processor, longer response times are expected. Although many enterprise IT environments have been implemented on machines with less than 3300 CPW with acceptable response times, the 3300 CPW processor has the possibility to achieve better overall performance. As with other platforms, the best performance and response times come from the latest and fastest processor technology.

This held true in the tests discussed in this paper. Faster response times occurred with the whole environment running on a POWER5 machine (two-tier environment) than with the HTTP and WebSphere Application Server being moved to a slower POWER3 machine (three-tier environment).

Partial-processor logical partitions

Running on partitions with less than one processor can also affect performance. For example, when running on a 0.5-way system, the partition is allocated 50% of the processor, while the other 50% is doing something else. This can effect response times and capacity on a processor-intensive workload, such as Web-facing.

To analyze this phenomenon, three configurations were run in the three-tier environment, with three capped processor configurations: 0.5-Way, 1-Way, and 2-Way. For each test, the number of virtual users was capped at 50. The results are displayed in Table 7:

Number of processors on HTTP/WebSphere Application Server machine	Number of users	Response time (seconds)
0.5 processors	50	0.17
1.0 Processors	50	0.076
2.0 Processors	50	0.064

Table 7: Three-tier environment

Note that the response time of the 0.5 processor is more than double that of the 1.0 processor. This is expected because 50% of the processor is allocated elsewhere on a 0.5 way system. If a CPU-intensive application, such as the IBM WebFacing Tool, is running on a partial processor, you can anticipate longer response times. The easiest way to avoid this is to run with at least one processor.

Memory-constrained environments

Environments where the WebSphere Application Server runs in a pool with a limited amount of memory (between 768 megabytes and 1 gigabyte per instance) are the most common constrained environments.

Java and WebSphere applications are sensitive to the amount of memory available to the JVM. To run optimally, Java workloads typically require more memory than traditional workloads. If a Java workload does not have adequate memory, the response time will often degrade until the JVM is restarted. To determine if an application has reached this state, perform the **Work with System Status** command (WRKSYSSTS), and look at the paging rates in the memory pool where the JVM is running. If the JVM is the only job running in the pool, and the workload has been adequately warmed up, rates fewer than 50 non-database pages per second will be typical (although rates can vary depending on the hardware configuration). In the demonstration shown in figure 20, only 1.5 non-database pages are brought into memory per second.

```

Work with System Status                                GOLLOM
                                                    10/23/05 16:36:12
% CPU used . . . . . :      44.2   Auxiliary storage:
% DB capability . . . . . :        .0   System ASP . . . . . :    175.4 G
Elapsed time . . . . . :    00:00:57   % system ASP used . . . :    85.9246
Jobs in system . . . . . :     5244   Total . . . . . :      175.4 G
% perm addresses . . . . . :    .009   Current unprotect used :    4486 M
% temp addresses . . . . . :    .105   Maximum unprotect . . . :    4866 M

Type changes (if allowed), press Enter.

System Pool Reserved Max ----DB---- ---Non-DB---
Pool Size (M) Size (M) Active Fault Pages Fault Pages
1 1093.90 346.85 +++++ .0 .0 .1 .1
2 11064.15 2.04 32000 .0 .0 1.2 2.8
3 3906.25 .23 1001 .0 .0 .4 1.5
    
```

Figure 20: Check paging rates with WRKSYSSTS

Likewise, disk usage can be monitored with the Work with Disk Status command (WRKDSKSTS) by examining the percent busy column. This is directly related to the previous factor. If paging rates are high, the disk is being accessed more often for pages of data. Thus, high paging rates lead to high disk usage.

Excessive paging rates and high disk utilization are indicators that the amount of memory is not adequate for the environment, resulting in the Java heap growing larger than the available memory. As the GC runs, it needs to access the entire heap to determine which objects are available for collection. If the heap is being paged out due to an inadequate amount of memory, the GC will have to page in the contents of the heap, resulting in disk usage and page faults.

As the number of JVMs concurrently running increases, more memory is needed to run them. If the first JVM requires 512 megabytes of memory, and the second JVM requires 768 megabytes of memory, then the combined total of 1.25 gigabytes will be required. To determine the approximate amount of memory required for a single JVM:

1. Start the JVM.
2. Ramp up the workload running on the JVM.
3. Perform the DMPJVM CL command on the running JVM and look at the output section labeled **Garbage Collection**.

Generally, the JVM will require the amount of memory indicated by the following formula, where JIT stands for Just-In-Time:

$$\text{heap size} + ((\text{JIT heap size} + \text{JVM heap size})/2)$$

Figure 21 shows an example of DMPJVM running against a live JVM.

```

.....
. Garbage Collection
.....
Garbage collector parameters
  Initial size: 98304 K
  Max size: 240000000 K
Current values
  Heap size: 636352 K
  Garbage collections: 3789
Additional values
  JIT heap size: 446720 K
  JVM heap size: 371928 K
  Last GC cycle time: 1134 ms
    
```

Figure 21: Sample DMPJVM running against a live JVM

The JVM in this sample needs an approximate minimum of 1 gigabyte of memory. This was calculated using the following formula:

$$636,352 + ((446,720 + 371,928)/2) = 1,045,676 \text{ kilobytes.}$$

Before using the JVM memory algorithm, ensure that enough memory is available to the JVM when DMPJVM is run. Otherwise, paging might cause the JVM heap to be artificially large, and that will skew the calculation.

After all tuning was complete on the application server, the amount of memory was reduced in the pool to 768 megabytes. Immediately, a problem occurred when the heap growing larger than the amount of memory available. After reducing the initial heap size from 256 megabytes to 96, the performance improved substantially. Even with the heap size set to 256 megabytes, As long as one gigabyte of memory is allocated to the pool, excellent performance can still be achieved. Another general rule is that at least four times the amount of memory set for the initial heap size is required in the pool.

The initial heap size can be set to minimize the amount of memory required. Lowering the initial heap size will reduce the amount of memory needed, but more CPU will be required. For more information about the initial heap size, refer to the **Resources** section at the end of this paper.

Disk-arm-constrained environments

After the JVM has started and loaded all classes for the application, the disk arms usually do not represent a critical performance factor. However, Java applications that have high database activity and those that write excessively to the iSeries integrated file system (IFS) can result in a bottleneck.

Although it is not discussed in this paper, the best way to view peak disk-arm utilization is to use collection services. However, to estimate quickly whether or not disk utilization is a problem, do the following:

1. Start the JVM.
2. Start the workload and produce stress on the JVM for a period of 10 minutes.
3. Perform a WRKDSKSTS command and press F10 every 5 to 10 seconds.
4. Look at the percent busy column. If a single disk arm is at 50% busy or higher during the period, the disk arms are in a bottleneck.

Note: A side effect of not having enough memory is high disk utilization. Therefore, it is important to eliminate the memory factor before investigating the disk arms.

Start-up and first touch optimizations

The following optimizations affect either the start-up time of the application or the first touch response time. These optimizations have no effect on runtime performance, however. Start-up refers to the time it takes for the IBM WebFacing Tool application to start within WebSphere Application Server. First touch is the response time received when each JSP is initially called, and it is usually slower than subsequent calls. The following optimizations are designed to reduce start-up and first touch times.

JSP batch compiler

Two JSP batch compilers are available with WebSphere Application Server. The first JSP batch compiler is a feature of the WebSphere Application Server administrative graphical user interface (GUI). The second is a script JSP batch compiler invoked from the command line in QSHHELL.

When JSPs in an application are compiled during installation, there will not be a first touch delay, as no compilation is needed at runtime. Depending on the number of JSPs in the application, and the server capacity, pre-compiling all JSPs can require a significant amount of time.

The GUI JSP batch compiler is a synchronous operation at application installation time. This means that the installation process will not complete until all JSPs are compiled. This is a good option for small applications, but it can cause extremely long installation times for large applications.

Figure 22 shows the JSP compiler option that can be checked during application installation.

Install New Application ⓘ

Provide Options to Perform the Install

Specify application deployment options

Application name: ⓘ

Directory application installed to: /QIBM/UserData/WebSphere/AppServer/V6/Base/profiles/

Pre-compile JSPs ⓘ

Note: Pre-compiling JSPs can have significant performance impacts. When enabled, the JSPs are compiled at installation time, causing the application install to take longer. When disabled, the JSPs are compiled the first time they are accessed, causing the first use of the application to take longer.

Figure 22: Pre-compile option

After the IBM WebFACING Tool application is installed into WebSphere Application Server, the script JSP batch compiler can be used. For more information on JSP batch compiler for WebSphere Application Server - Express V6, refer to the **Resources** section of this paper.

JSP pretouch tool

The JSP pretouch tool is designed to remove the first touch delay for users. To enable the JSP pretouch, three possible attributes can be configured. Each setting resides in the `ibm-web-ext.xmi` file of a Web module. Setting the parameters in this file can be done either during development or after deployment.

The two most important attributes are `prepareJSPs` and `prepareJSPAttribute`. The third, `prepareJSPThreadCount`, can be used on multiprocessor systems. A complete description of all three can be found in the iSeries Information Center, found in the **Resources** section of this paper.

Configuring pretouch attributes

The pretouch attributes can be configured during development in WebSphere Development Studio Client for iSeries, or by using a text editor to edit the deployed `ibm-web-ext.xmi` file.

Using WebSphere Development Studio Client for iSeries in the navigator view of the IBM WebFACING Tool project, `ibm-web-ext.xmi` can be found under...

WebContent > WEB-INF

Figure 23 shows an example of what the WebSphere Development Studio Client looks like when set with the **prepare** attributes having been set from the IBM WebFACING Tool.

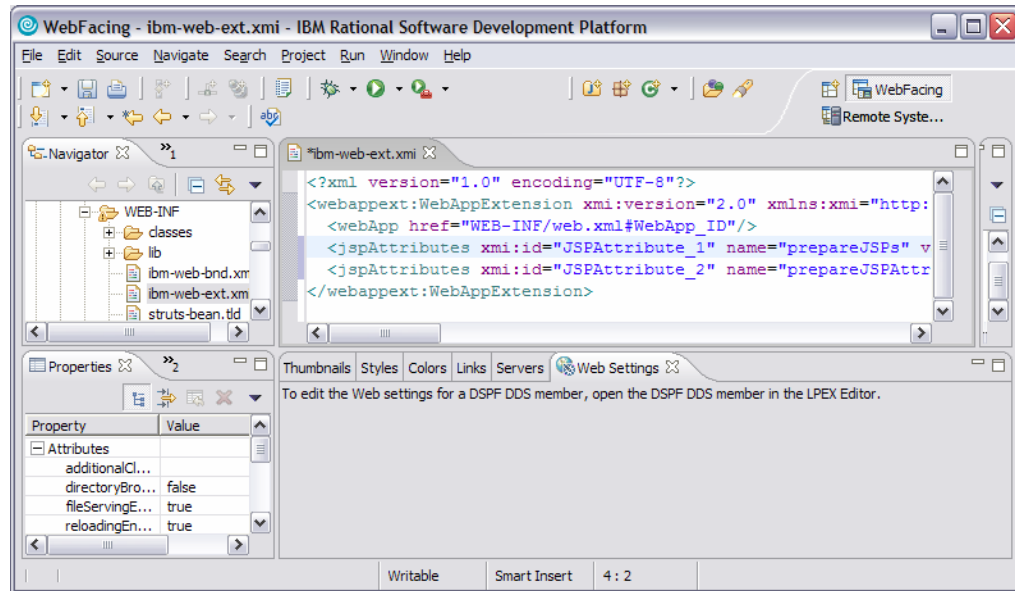


Figure 23: **Prepare** attributes set for the IBM WebFACING Tool

In a deployed application, the `ibm-web-ext.xmi` file can be edited to add or remove the JSP pretouch attributes after deployment. The `ibm-web-ext.xmi` file resides in two places: the deployed application directory and the cells directory. When modifying the file manually, the file must be modified under the cells directory path for the deployed application. The following is the location of the deployed location of the XML file where attributes can be added or removed for WebSphere Application Server Base V6:

```
\QIBM\UserData\WebSphere\AppServer\V6\Base\profiles\

```

The resulting `ibm-web-ext.xmi` file settings created from the WebSphere Development Studio Client Tools to edit the `ibm-web-ext.xmi` file appear as follows:

```
<?xml version="1.0" encoding="UTF-8"?>

<webappext:WebAppExtension xmi:version="2.0"
xmlns:xmi="http://www.omg.org/XMI" xmlns:webappext="webappext.xmi"
xmi:id="WebAppExtension_1131646333235" reloadInterval="3"
reloadingEnabled="true" additionalClassPath="" fileServingEnabled="true"
directoryBrowsingEnabled="false" serveServletsByClassnameEnabled="true">

  <webApp href="WEB-INF/web.xml#WebApp_ID" />

  <jspAttributes xmi:id="JSPAttribute_1" name="prepareJSPs" value="0"/>

  <jspAttributes xmi:id="JSPAttribute_2" name="prepareJSPAttribute"
value="JunkEntryPoint"/>

</webappext:WebAppExtension>
```

During compilation, the JSP pre-touch tool writes information to the `SystemOut.log` file, indicating progress was made. When all JSPs have finished compiling, a final message indicating such will exist in the log. A sample from the `SystemOut.log` file showing that all 656 JSP files have been processed appears in Figure 24 below.

```
jsp      I   PrepareJspHelper in group [Flight400]: 575 jsp files have been processed.
jsp      I   PrepareJspHelper in group [Flight400]: 600 jsp files have been processed.
jsp      I   PrepareJspHelper in group [Flight400]: 625 jsp files have been processed.
jsp      I   PrepareJspHelper in group [Flight400]: 650 jsp files have been processed.
jsp      I   PrepareJspHelper in group [Flight400]: All 656 jsp files have been processed.
```

Figure 24: Sample segment of `SystemOut.log` file

Once all JSP files have been compiled, these attributes can be removed from the deployed application so that on subsequent server restarts the WebSphere Application Server will skip the step of checking all JSP files for recompilation.

Bytecode cache

The bytecode cache, which is only available on an iSeries system, is designed to improve the startup time of the application. The cache is an empty jar file that contains Java programs that can improve the startup performance of classes loaded by user class loaders. This is done by allowing the Java program objects (JVAPGMs) created by user classloaders to be cached for reuse, avoiding JVAPGM creation and bytecode verification during the initial class load. WebSphere components are loaded by user class loaders, and they can take advantage of this feature.

Using bytecode caching might be useful if there are issues during application server startup and termination time, and component runtime during first touch of a JSP.

The user class loader cache can improve performance in two ways. The first way is avoiding bytecode verification. If the program is already in the cache, bytecode verification is not performed again. The second is by avoiding the temporary creation

of JVAPGMs. If the program is already in the cache, the existing JVAPGM can be used. Because any optimization level can be stored in the cache, it is more practical to consider higher optimization levels.

In both cases, the first time the class is loaded (for example, before the cache is primed or after a class is changed), these functions are performed, and the load is slower. Bytecode verification is permanent when using the cache file. Once the class is already in the cache, subsequent loads are much quicker.

Using the user classloader cache

To enable the user class loader cache, the Java system property `os400.define.class.cache` file must be set. This setting is changed in the **Custom Properties** panel of the WebSphere Application Server administrative console. The `os400.define.class.cache` setting specifies the full path name of a valid JAR file that holds the Java program objects. This JAR file must contain a valid JAR entry.

You can find details on the user classloader cache in the WebSphere Application Server V6 Information Center, which is listed in the **Resources** section of this paper. Information on other versions of WebSphere Application Served can be found at their respective IBM Information Centers.

Summary

To recap the settings discussed in this paper, Table 8 explains the changes that were done to improve the amount of CPU being consumed by the WebSphere job.

Setting	Pros	Cons
Edge Side Includes (ESI) cache	Automatically caches static object types (GIF, JPEGs HTML) served from WebSphere in the IBM HTTP Server plug-in	Only works if using IBM HTTP Server in front of WebSphere
Setting initial heap size to a larger value	Can reduce CPU consumed by JVM (Garbage Collector in particular)	<ul style="list-style-type: none"> Is a trade off between memory and CPU Larger the value, more memory is required.
Setting java.compiler=jitc	<ul style="list-style-type: none"> Can reduce CPU consumed by JVM Is faster than direct execution mode in V5R3 	<ul style="list-style-type: none"> Can result in slow startup times on lower end systems Can negatively affect performance on V5R3 and prior
Turning compression off in the IBM WebFACING Tool environment	Can reduce CPU consumed, due to avoiding the compressing the outgoing HTML	On limited bandwidth environments, compression can reduce the impact on slow network speeds.
Record definition cache	Can reduce CPU and memory consumed by caching record definitions	<ul style="list-style-type: none"> Too low of a setting degrades performance Too high of a setting wastes memory

Table 8: Performance settings summary

Table 9 contains three settings that improve either the startup time or first-touch time.

Setting	Pros	Cons
JSP Pre-touch tool	Will load all JSPs on application server startup, making a first call to a JSP quicker	Will increase JVM startup times due to loading all of the JSPs when the application server starts
Bytecode verification cache	Will let the application server start up faster	Has no positive effect on runtime
JSP pre-compiler	Eliminates the dynamic JSP compile that has to be done once for the life of the application	Can cause excessive load times

Table 9: Three improvement settings

Table 10 lists a few things to try for constrained environments:

Environment	Things to try
Memory-constrained environment	Lower the initial heap size. Do not set the initial heap size below 96 megabytes.
Processor-constrained environment	If there are logical partitions, consider dedicating at least one full processor to the JVM running WebSphere. Anything less will impact response times.
Disk-arm-constrained Environment	Disk-arm-constrained environments typically only affect startup times in a Web-facing environment, unless the back-end native jobs require more
Other workloads running on partition	<ul style="list-style-type: none"> Run WebSphere subsystem in a separate pool. Adjust the priority of the Web-facing jobs accordingly.

Table 10: Tuning options for memory-constrained environments

Resources

These Web sites provide references to supplement the information contained in this document:

- WebSphere Application Server for iSeries Web site:
ibm.com/servers/eserver/series/software/websphere/wsappserver/
- WebSphere Application Server for OS/400 V6.0 Information Center:
<http://publib.boulder.ibm.com/infocenter/wsdoc400/index.jsp>
- Basic Java Performance for iSeries (bytecode verification):
ibm.com/servers/eserver/series/perfmgmt/pdf/BasicJavaPerf.pdf
- Tuning Garbage Collection for Java and WebSphere on iSeries:
ibm.com/servers/eserver/series/perfmgmt/pdf/tuninggc.pdf
- WebSphere Application Server Group PTFs:
ibm.com/servers/eserver/series/software/websphere/wsappserver/services/service.html
- Redbook: Maximum Performance with WebSphere Application Server V5.1 on iSeries:
www.redbooks.ibm.com/redbooks
- IBM WebFacing Tool Performance Update using WebSphere Application Server V5.0:
ibm.com/servers/enable/site/education/abstracts/webappv5_abs.html
- IBM WebFacing Tool Roadmap:
www.developer.ibm.com/vic/hardware/myportal/develop/roadmap?roadMapId=appinita
- Virtual Innovation Center for Hardware IBM WebFacing Tool site:
ibm.com/servers/enable/site/ebiz/webfacing/start.html
- iSeries Information Center Version 5 Release 3
<http://publib.boulder.ibm.com/infocenter/series/v5r3/index.jsp?topic=/rzatz/51/program/jspprtch.htm>
- JSP batch compilation in WebSphere Application Server Version 6
http://publib.boulder.ibm.com/infocenter/wsdoc400/index.jsp?topic=/com.ibm.websphere.iseries.doc/info/ae/ae/cweb_jspbchtl.html
- IBM eServer iSeries support fixes:
ibm.com/servers/eserver/support/series/fixes/index.html
- Paragon Consulting Services, Inc.:
www.paragon-csi.com

About the authors

Scott Moore is a Senior Performance Analyst in the IBM Rochester lab and an OS/400 and Java expert. Recently, Scott has been the team leader for WebSphere and Java performance on iSeries, and is currently helping solution providers with Java and WebSphere performance through the IBM eServer Solutions Enablement team.

Michael Sandberg is a technical consultant in the IBM eServer Solutions Enablement team, located in Rochester, Minnesota. For the past five years, he has been involved in supporting iSeries solution providers as they enhance and innovate their applications. As part of this work, he has accumulated technical expertise in the IBM WebFacing Tool, Host Access Transformation Services (HATS), and other IBM technologies that focus on iSeries application innovation.

Trademarks and special notices

© IBM Corporation 1994-2005. All rights reserved.

References in this document to IBM products or services do not imply that IBM intends to make them available in every country.

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

IBM	eServer	i5/OS	DB2
ibm.com	iSeries	WebSphere	DB2 Universal Database
the IBM logo	Rational	POWER	POWER4
Redbooks	PowerPC	POWER5	POWER3

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

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

Intel, Intel Inside (logos), MMX and Pentium are trademarks of Intel Corporation in the United States, other countries, or both.

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

Information is provided "AS IS" without warranty of any kind.

All customer examples described are presented as illustrations of how those customers have used IBM products and the results they may have achieved. Actual environmental costs and performance characteristics may vary by customer.

Performance is based on measurements and projections using standard IBM benchmarks in a controlled environment. The actual throughput or performance that any user will experience will vary depending upon considerations such as the amount of multiprogramming in the user's job stream, the I/O configuration, the storage configuration, and the workload processed. Therefore, no assurance can be given that an individual user will achieve throughput or performance improvements equivalent to the ratios stated here.