WebSphere™ Application Server 3.5 for iSeries
Performance Considerations

Jill Peterson
Eric Barsness
IBM Rochester Lab

November 2000

This document is an update to the "WebSphere Application Server 3.0.2 for AS/400 Performance Considerations" document[1]. The concepts and data included in the "3.0.2 Performance Considerations" document generally still apply with version 3.5, however some of the details may have changed.

WebSphere is but one component of many in a production environment. The client, the communications network, the web server, the application server, the database server, and the Java™ Virtual Machine (JVM) all play a role in performance.

Differences from WebSphere Application Server 3.0.2

The property for setting the Prepared Statement Cache Size is specified differently in WebSphere 3.5. This was one of the more significant performance properties in WebSphere 3.0.2 and proves to be in WebSphere 3.5 as well. In WebSphere 3.0.2 the property was associated with the Application Server and was set in the General tab Command Line Argument field by adding -Dcom.ibm.ejs.dbm.PrepStmtCacheSize=1000. Now, the Prepared Statement Cache is associated with each data source. To specify the property, you need to create a datasources.xml file and put it in your [Instance Root]/properties directory. For example, the default directory would be /QIBM/UserData/WebASAdv/default/properties. Here is an example of what the datasources.xml file should look like:

```
<data-sources>
<data-source name=”Default DataSource”>
<attribute name=”statementCacheSize” value=”1000”/>
</data-source>
</data-sources>
```

Another setting that you might see differently in WebSphere 3.5 is the syntax for setting JVM Parameters in your Application Server Command Line Argument field. For example, in WebSphere 3.0.2, JDK1.1.8 required that you set the initial garbage collection heap size by adding -ms256m. In WebSphere 3.5 , JDK1.2.2 requires that you use -Xms256m. WebSphere 3.5 for iSeries will accept the setting either way. In general WebSphere 3.5 requires a larger initial heap size than WebSphere 3.0.2 did.

In WebSphere 3.0.2 one of our recommendations was to use the system classloader for user code and running with DE optimization level 40. The JIT has improved enough with V4R5 and the user code is usually such a small portion of what is running that the difference is negligible. In WebSphere 3.5 it is preferred to use the user classloader in order to take advantage of the reloadability features. This is the default behavior.

If you have more than 50 unique URLs actively being used through WebSphere, you might want to increase the Invocation Cache parameter. The Invocation Cache holds mapping information between request URLs and servlet resources. The size of the cache can be specified for your application server along with other JDK parameters in the Command Line Arguments field by using: -DinvocationCacheSize=50. A cache of the specified size will be created for each thread/process, as determined by the servlet engine MaxConnections setting. One caution with increasing this cache size is that it uses more space in the Java heap.

WebSphere administration tracks a variety of configuration information about WebSphere resources. Some of this information needs to be understood by the web server as well, such as URIs pointing to WebSphere resources. This configuration data is pushed to the web server via the WebSphere plug-in. This allows new servlet definitions to be added without having to restart any of the WebSphere servers. Unfortunately, the dynamic regeneration of this configuration information is an expensive operation. There is a setting in [websphereroot]/properties/bootstrap.properties to change the refresh interval between these updates. The property is ose.refresh.interval. The default value for ose.refresh.interval is 10. Changing this number to a higher value may improve performance.

In WebSphere 3.02 and above, the JSP processing engine is implemented as a servlet. The JSP 1.0 servlet is invoked to handle requests destined to all JSP files matching the assigned URL filter, such as /myApp/*.jsp. Each time a JSP file is requested, the corresponding disk file is checked to see if a newer version is available. This can be an expensive operation. Change the interval by adding an Init parameter, minTimeBetweenStat, to the JSP Servlet. (Node : Application Server : Servlet Engine : Web Application : JSP 1.0 Processor : Advanced Tab : Init Parameters) The default value is 1000. The recommended value is 100000.

Something else to note, the Adminserver startup will have a slightly higher response time in WebSphere 3.5 than it did in WebSphere 3.0.2 because of additional function that was added.

Performance Updates in WebSphere 3.5.1

Prior to v3.5.1 when getSession( ) was called in a Servlet/JSP, this would lock the session until the service( ) method completed. This is okay if only one Servlet/JSP ever wants to use the session, but other Servlets/JSPs also issuing a getSession( ) would be blocked in the meantime. At v3.5.1, there was a change whereby locking of the session is no longer performed. This is especially important when you employ frames at your browser. Each frame represents a potential JSP, Servlet, HTML, gif, etc that execute concurrent HTTP requests. Previously, the first Servlet of a frame to issue a getSession( ) would get the lock on the session, and Servlets/JSPs in the remaining frames had to wait serially for their getSession( ) to be freed.

Session Affinity, which did not exist in WebSphere 3.0.2, is on by default in WebSphere3.5.1 and improves performance in a cloned environment with session.

Performance Updates in WebSphere 3.5.2

Prior to v3.5.2, there was a hard-coded maximum = 100 for the number of worker threads at the application server. This was controlled by "Max Connections" at the Advanced tab of the Servlet Engine, whose default at v3.5x is set to 25. If this value is set to 400 for example, 400 worker threads would be created, however only up to 100 threads would actually be utilized concurrently, and 300 would be idle. At v3.5.2 this hard-coded limit was lifted.

Performance Recommendations

Resource Analyzer

The Resource Analyzer is a stand alone application in WebSphere 3.5 rather than an integrated part of the console as it was in WebSphere 3.0.2. It can be used to monitor many performance-related values in WebSphere. While the Resource Analyzer will not be discussed further in this document, it can be used to help determine the correct settings for many of the following performance-related considerations. See the WebSphere Application Server Version 3.5 Documentation Center[2] for more information about the Resource Analyzer.

General Java Performance

- Apply the most recent Java group PTF. Several Java PTFs have been made available for V4R4 and V4R5, which may have a performance impact. Also be sure to apply the latest WebSphere group PTFs.
- If you have many active threads, ensure that the activity level is set high enough in the storage pool in which you are running in. To modify the activity level for your pool, use the Work with System Status (WRKSYSSTS ASTLVL(*INTERMED)) command to change the value in the Max Active column. In addition, the system value QMAXACTLVL should be set to a value equal to or greater than the total activity level for all pools, or *NOMAX (use WRKSYSVAL SYSVAL(QMAXACTLVL)). Increasing these values should reduce or eliminate threads transitioning into the ineligible condition. More information on activity level can be found in the *OS/400 Work Management* guide[3] .
- Systems that have AnyNet support enabled may see worse performance when running WebSphere applications than those that do not have AnyNet support enabled. By default, AnyNet support is not enabled. To check if AnyNet support is enabled on your system, use the DSPNETA command. To disable AnyNet support (if it is not used by other applications), use CHGNETA ALWANYNET(*NO).

The section titles below indicate where the properties discussed are specified in the Administration Client GUI. For example, the application server settings are specified on the GUI client under the Application Server under the Node on the Topology tab.

Application Servers (Node : Application Server)

- If using the JSP 0.91 specification, use the JSP 0.91 batch compiler to enable faster responses to the initial client requests for JSP files. There is a batch compiler for the JSP 1.0 specification in WebSphere 3.5.2. See the IBM WebSphere Application Server Version 3.5 Documentation Center[2] for more information.
- Tune the initial garbage collection heap size. This value is initially set to 32 MB, but a larger value may improve performance. (General : Command Line Arguments : -Xms32m)
- Do not enable tracing unless required for debugging. (Advanced : Trace Specification and Debug : Debug enabled, Object Level Tracing enabled)
- Leave the ping interval at its default value. (Advanced : Ping Interval)
- Increasing the ping timeout and ping initial timeout values may help avoid unnecessary restarts of the application server on heavily loaded or very slow systems. Do not modify these values unless the application server is being restarted unnecessarily. (Advanced : Ping Timeout, Ping Initial Timeout)
- On AS/400, the process priority value is ignored. (Advanced : Process Priority)
- The thread pool size value is ignored in WebSphere version 3. (Advanced : Thread Pool Size)

Servlet Engines (Node : Application Server : Servlet Engine)

- Leave OSE as the default queue type. (Advanced : Queue Type)
- Leave Local Pipes as the default transport type when using the OSE queue type. (Advanced : Settings : Transport Type)
- Tune the maximum connections for the transport type. This value should be equal to or slightly less than the MaxActiveThreads value for the web server. Information about the MaxActiveThreads value can be found in the *AS/400 Performance Capabilities Reference*[5]. (Advanced : Max Connections)

Web Applications (Node : Application Server : Servlet Engine : Web Application)

- Disable automatic reloading of servlets. (Advanced : Auto Reload)
- If automatic reloading is necessary, set the reload interval as high as can be tolerated. (Advanced : Reload Interval)
- Do not enable context sharing if it is not required. (Advanced : Use Shared Context)

Session Management (Node : Application Server : Servlet Engine : Session Manager)

- Do not enable persistent sessions unless required by the application. (Enable : Enable Persistent Sessions)
- If persistent sessions are required, enable multi-row sessions and caching. (Tuning : Using Multi-row Sessions, Using Cache)
- Set the base memory value to a number close to the average number of active sessions. If there are frequent periods where more sessions are active, increasing the base memory value further may improve performance. (Tuning : Base Memory Size)
- If possible, invalidate finished sessions promptly in your application. This can be done by using the javax.servlet.http.HttpSession.invalidate() method.

JDBC Drivers and DataSources

- Use the native JDBC driver (com.ibm.db2.jdbc.app.DB2Driver). (JDBC Driver : Class Name)
- Only enable JTA for the JDBC driver if it is required by the application. (JDBC Driver : JTA Enabled)

- Increase the number of active connections if necessary, but try to keep the value as close to the required number of concurrent connections as possible. (DataSource : Advanced : Maximum connection pool size)
- Adjust the Minimum connection pool size to a value close to the average minimum number of concurrent connections required by the application. (DataSource : Advanced : Minimum connection pool size)
- Leave the connection, idle, and orphan timeout values at the default settings. (DataSource : Advanced : Connection timeout, Idle timeout, Orphan timeout)
- Adjust the QSQSRVR prestart job settings for your system. On AS/400, Java database access (JDBC) occurs via QSQSRVR jobs. By default there are five of these jobs initially active. Also by default, when less that two QSQSRVR jobs are unused, two more are created. An application that establishes a large number of database connections over a short period of time may be able to more quickly create those connections if these values are increased. Use the CHGPJE SBSD(QSYS/QSYSWRK) PGM(QSYS/QSQSRVR) command and increase the initial number of jobs, threshold, and additional number of jobs values. Only start the number of jobs required by the application, as there is some overhead associated with having QSQSRVR jobs active, even if they are not being used.
- Use the JDBC 2.0 connection pooling APIs in your application. See the online documentation for more information about connection pooling and its benefits (Data Access : With Servlets in AS/400 WebSphere Application Server documentation).

Security (Console : Tasks)

- Only turn on security if it is required for the application. (Configure Global Settings : Enable Security)
- If security is enabled, tune the security cache timeout for your environment. The larger the value, the better the performance. (Configure Global Settings : Security Cache Timeout)

EJB Container (Node : Application Server : EJBContainer)

Advanced Edition only

- Tune the cache settings (absolute and preferred limits, size, cleanup interval) to avoid passivation of objects. To estimate the required value for the absolute limit property, multiply the number of entity beans active in any given transaction by the total number of concurrent transactions expected. Then add the number of active session bean instances. (Advanced)
- Set the methods on your EJBs to read-only whenever possible. This can be done in VisualAge for Java™ 3.5 through the console. (EnterpriseBean : Edit Deployment Descriptor : Transactions : Read-Only)
- Use the lowest transaction isolation level required by your application. This will result in less database locking. Do not use TRANSACTION_SERIALIZABLE unless absolutely necessary. (EnterpriseBean : Edit Deployment Descriptor : Isolation : Isolation Level)
- Set the database access on your beans to exclusive whenever possible. Note, you can only do this if your WebSphere applicatoin is the only application accessing the data. Otherwise, you must use Shared. (EnterpriseBean : General : Database Access)
- Cache home references in your application. If you use VisualAge for Java™ 3.5, access beans do this for you automatically. To work with a bean, you first have to get the EJB Home object, and then use it to create individual Bean objects. Home objects represent the type of bean, while Bean objects represent a specific instance. Therefore, while Bean objects have to be retrieved for each transaction, Home objects can be shared from one transaction to the next, and by multiple users. Retrieving a reference to a Home object requires getting the InitialContext, invoking its lookup() method, and casting the resulting bean to the proper type. Eliminating the repetition of these steps can yield a performance improvement, especially with short transactions, or transactions

that involve multiple beans. To eliminate these steps, you can include a static field in your client class to hold a reference to the Home object, such as:

```
private static CustomerHome customerHome = null;
```

Then, at some point before you use the home object, you can retrieve it as usual:

```
if (customerHome == null) {
/* lookup the Home object as usual */
...
}
```

After this code is executed once, no further Home lookups should be necessary.

General EJB Pointers (not specific to iSeries)

- In your application development start with a small number of EJBs. In general your application will work better and be easier to maintain and understand with a smaller number of beans.
- Use entity beans to represent large coarse objects. For example use an entity bean to represent an entire order, not each individual order line. Use entity beans to represent data and session beans to represent the logic of your application.

Capacity Planning Information

There are no easy answers to capacity planning questions. Use the IBM Workload Estimator for iSeries 400[6] for sizing WebSphere. Consult Chapter 22 of the *AS/400 Performance Capabilities Reference Version 4 Release 5*[5] for for more information on the Workload Estimator, Chapter 6 for more information on Webserving Performance capacity planning information, or Chapter 7 for general Java Capacity planning information on AS/400.

We collected some data on an internal workload to compare the performance between WebSphere3.0.2 and WebSphere3.5. Our particular workload takes advantage of servlets, JSPs and EJBs.

System Information	Increase in Throughput
830-2402 Condor 4-way	6%
170-2385 Northstar 1-way	2%

Additional Information

WebSphere is but one component of many in a production environment. The client, the communications network, the web server, the application server, the database server, and the Java Virtual Machine (JVM) all play a role in performance. The reader may want to review the following documents to understand the performance role of these many components:

See *AS/400® Performance Capabilities Reference Version 4 Release 5*[5] for information on tuning these components.

"Chapter 4. DB2 UDB for AS/400 Performance"
"Chapter 5. Communications Performance" (see for TCP/IP tuning)
"Chapter 6. Web Serving Performance"(see for HTTP server configuration tuning)
"Chapter 7. Java Performance"

Building AS/400 Applications with Java Redbook[7] and the following chapters:

"Chapter 13. Java Performance and Work Management Overview"
"Chapter 14. Java Application Design and Host Performance Analysis"
"Chapter 15. Application Performance Analysis with GUI"

This document does not address the performance effects of cloning. It also does not discuss the function available in, or how to use, the Resource Analyzer. For more information on these topics, please reference the Application Server Version 3.5 Documentation Center[2].

The suggestions in this document are intended for use in production application server environments. When working in a development or test environment, suggestions such as using the system classloader or disabling automatic reloading of servlets may not be productive. Keep this in mind when considering the information presented.

Performance data in this document was obtained in a controlled environment with specific performance benchmarks and tools. This information, and general recommendations, is intended to give the reader a better understanding of IBM products. Results obtained in other environments may vary significantly and do not predict a specific customer's environment. Some testing was done on non-AS/400 versions of WebSphere. However, the concepts should apply to AS/400.

The information contained in this document has not been submitted to any formal IBM test and is distributed "as is" without warranty of any kind, expressed or implied. IBM expressly disclaims the implied warranties of merchantability, fitness for a particular purpose, or non-infringement. The use of this information or the implementation of any of these techniques is a customer responsibility and depends on the customer's ability to evaluate and integrate them into the customer's operational environment. While each item may have been reviewed by IBM for accuracy in a specific situation, there is no guarantee that the same or similar results will be obtained elsewhere. Customers attempting to adapt these techniques to their own environments do so at their own risk.

Useful Links

"WebSphere Application Server 2.02 Performance Considerations"
http://www.as400.ibm.com/tstudio/websphere/product/WSA202PerformanceConsiderations.html

WebSphere Application Server Development Best Practices for Performance and Scalability
http://www-4.ibm.com/software/webservers/appserv/ws_bestpractices.pdf

VADD-Choosing the Right EJB Type: Some Design Criteria
Http://www7.software.ibm.com/vad.nsf/data/document2361?OpenDocument&p=1&BCT=1&Footer=1

VADD-Design and Implement Servets/JSPs/EJBs for IBM WebSphere
Http://www7.software.ibm.com/vad.nsf/Data/document1931?OpenDocument&p=1&BCT=1&Footer=1

VADD-WebSphere Scalability: WLM and Clustering Using WebSphere Application Server Advanced Edition
Http://www7.software.ibm.com/vad.nsf/Data/document4321?OpenDocument&p=1&BCT=1&Footer=1

VADD-EJBs and Transaction Management in WebSphere Advanced Edition
Http://www7.software.ibm.com/vad.nsf/Data/document2357?OpenDocument&p=1&BCT=3&Footer=1

Additional Support and Author Contacts

IBM Global Services at Rochester, MN, offers technical support for Performance Analysis & Capacity Planning of AS/400 products. Their services are described on the internet at: http://www.as400.ibm.com/service/igs/pss.htm

Questions regarding the conclusion reached in this white paper should be addressed to Jill Peterson, jillpete@us.ibm.com , or Richard Odell, rjodell@us.ibm.com .

Trademarks

The following terms are trademarks of the IBM Corporation in the United States or other coutries or both:

AS/400, IBM, VisualAge, and WebSphere are trademarks of International Business Machines Corporation in the United States, other countries, or both.

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

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

References

[1]*"WebSphere Application Server 3.0.2 for AS/400 Performance Considerations"*
http://www.as400.ibm.com/products/websphere/product/performanceAE302.html

[2]IBM WebSphere Application Server Version 3.5 Documentation Center
http://www.as400.ibm.com/products/websphere/docs/doc.htm

[3]*OS/400 Work Management V4R4*
http://publib.boulder.ibm.com:80/cgi-bin/bookmgr/BOOKS/QB3ALG03/CCONTENTS

[4]*AS/400 Developer Kit for Java*
http://publib.boulder.ibm.com/pubs/html/as400/v4r4/ic2924/info/java/rzaha/devkit.htm

[5]*AS/400 Performance Capabilities Reference Version 4, Release 5*
Http://publib.boulder.ibm.com/pubs/html/as400/online/chgfrm.htm

[5]*AS/400 Performance Capabilities Reference Version 4, Release 4*
http://publib.boulder.ibm.com/pubs/html/as400/online/chgfrm.htm

[6]IBM Workload Estimator for iSeries 400
http://as400service.ibm.com/estimator

[7]*Building AS/400 Applications with Java*
http://www.redbooks.ibm.com/abstracts/sg242163.html