[October, 2009]

IBM WebSphere Application Server v7.0 Performance Optimization Scripts

Document version [1.0]

WebSphere software

Christopher J. Blythe

Introduction

A new set of optimized performance tuning scripts are now available for download on the WebSphere® Application Server Performance site and included within the latest update to the WebSphere Application Server Base Trial v7.0. These python-based tuning scripts can be used in conjunction with wsadmin to apply recommended tuning settings for three standard tuning templates. These standard tuning templates are designed to target individual server instances and apply some of the most common WebSphere Application Server tuning parameters to suit one of the following three environments:

- Production – Applies tuning well suited for a production environment where application changes are rare and optimal runtime performance is desired.
- Performance Test – This template is very similar to the production template; however, adjustments have been made to account for potential changes to the application and to generate diagnostic information like verbose garbage collection output.
- Development – Tunes the server for a development environment where frequent application updates are performed and system resources are at a minimum.

A fourth tuning script (Default) is also provided that will return the server configuration to the standard out-of-the-box defaults.

The tuning parameters and values applied by these templates may not result in optimal performance for your specific application. These templates should be viewed as a recommended starting point for improving application server performance. We highly recommend conducting your own performance evaluation and tuning exercise in order to fine tune the server for your application.

Running the Scripts

The tuning script packages are located in the following directory:

<WAS_HOME>/scriptLibraries/perfTuning/V70

In this directory you will find the following four python tuning scripts corresponding to the four templates discussed in the introduction.

- *production_template.py*
- *performanceTest_template.py*
- *development_template.py*
- *default_template.py*

To apply a tuning template to your server profile, execute the following command:

<WAS_HOME>/profiles/<PROFILE>/bin/wsadmin -f ../../../scriptLibraries/ perfTuning/V70/<script template>

If more than one server configuration is found in your profile, append the following script arguments to the command to identify the target node and server:

-node <node name> -server <server name>

An additional script, *view_parameters.py*, can be used to review the current values of server parameters modified by the tuning scripts. This script can be invoked using the same method described above.

The scripts can be applied to a server profile at any time. However, to benefit from the Data Source and class reload tuning described in the next section, the scripts should be executed after your application and its required resources have been installed.

Tuning Performed By Scripts

The following table details the specific tuning performed by each of the three scripts and compares this to the original out-of-the-box defaults. If one of the listed parameters is omitted or configured back to the server defaults for a particular tuning template, the cell will be left blank in the table. Further information regarding each tuning parameter and the impact it has on the server performance is discussed later.

Parameter	Server Default	Production	Performance Test	Development
JVM Heap Size (MB)	50 min / 256 max	512 min / 512 max	512 min / 512 max	256 min / 512 max
Verbose GC	disabled	disabled	enabled	enabled
JVM Diagnostic Trace (Generic JVM Arguments)		-Xtrace:none	-Xtrace:none	-Xtrace:none
HTTP (9080) and HTTPS (9443) Channel maxKeepAliveRequests	100	10000	10000	10000
Development Mode	disabled			enabled
Server Component Provisioning	disabled	enabled	enabled	enabled
PMI	enabled	disabled	disabled	disabled
Authentication Cache Timeout *	10 minutes	60 minutes	60 minutes	60 minutes
Override Application Class Reload Interval *	No – 3 sec (enabled)	Yes – 0 sec (disabled)	Yes – 60 sec (enabled)	
JSP Reload Interval	enabled – 10 sec	disabled	enabled – 60 sec	
Data Source Connection Pool Size *	1 min / 10 max	10 min / 50 max	10 min / 50 max	
Data Source Prepared Statement Cache Size*	10	50	50	
ORB Pass-by-Reference	disabled	enabled	enabled	enabled
Thread Pools (Web Container, ORB, Default)	50 min / 50 max, 10 min / 50 max, 20 min / 20 max			5 min / 10 max

*Indicates items that must exist in the configuration to be tuned. Any new items will be created using the standard server defaults

JVM Heap Size – The Java™ Virtual Machine (JVM) heap size is typically one of the first parameters tuned in any WebSphere Application Server environment. The overall goal of tuning the heap size is to not only reduce the frequency of garbage collections, but also minimize the duration of each garbage collection cycle. This maximizes the number of cycles granted to the server to perform application work. The default size of 50 MB minimum and 256 MB maximum is generally too small for most applications. Increasing the heap size to 512 MB minimum and maximum generally reduces the frequency of garbage collections and eliminates the overhead incurred by the JVM for dynamically managing the heap size. For the Development profile where optimal performance is not a key factor, the minimum is left at 256 MB to reduce the overall size of the JVM process.

Java virtual machine settings
Tuning the IBM virtual machine for Java
Tuning the HotSpot Java virtual machines (Solaris & HP-UX)

Verbose GC – The verbose garbage collection output generated by the JVM is an essential component in fine tuning the JVM heap size and garbage collection policy. When enabled, this information is written to the application server's native_stderr.log. There is very little performance overhead associated with verbose garbage collection; however, there is a risk that the log file could grow without bound if left unchecked. Consequently, the Production profile leaves verbose garbage collection disabled.

Java virtual machine settings

JVM Diagnostic Trace – The low-level JVM diagnostic trace facilities are rarely needed and can be disabled to eliminate associated overhead. Other diagnostic capabilities like the ability to generate javacores and heap dumps are not disrupted.

Maximum Keepalives Per Request – This parameter adjusts the number of requests a keepalive connection can service before the server forces the connection to be closed. This capability is used to prevent denial of service attacks; however, the default value of 100 is relatively small. Increasing this parameter has a significant impact on SSL connections where the initial SSL handshake is a costly operation.

HTTP transport channel settings

Development Mode – This feature reduces the time needed to start the application server by adding the –Xquickstart and -Xverify:none parameters to the server JVM command line. The quickstart option directs the JVM to perform class optimization at a lower level, providing an improvement in JVM startup time. The verify:none parameter directs the JVM to skip class verification which can also provide benefits to JVM startup time.

Application server settings

Server Component Provisioning – The feature can potentially reduce the startup time and memory footprint of the application server by allowing internal server components to start as needed. For

instance, if an application consists of Java Servlets and JavaServer Pages (JSP™) only, there is no need for the server to start the EJB container.

Application server settings

Performance Monitoring Infrastructure (PMI) – The PMI service is enabled by default and provides common performance statistics for the sever and installed applications. This service is generally useful for performance tuning and problem determination, but does add additional overhead. Consequently, we recommend disabling this service unless it is explicitly needed.

Enabling PMI data collection

Authentication Cache Timeout – The authentication cache timeout controls how often authentication information is refreshed within the cache. Obtaining authentication information from an LDAP server or other native authentication mechanisms is a costly operation. Increasing the authentication cache timeout reduces the frequency of these costly updates.

Authentication cache settings

Override Application Class Reload Interval – By default, the application server scans for application class changes every 3 seconds. This is good for development environments where application changes are expected on a frequent basis, but may not be necessary in other environments. For instance, in a test environment less frequent application changes are expected, the default value can be overridden and increased to 60 seconds. In a production environment, this feature can be disabled all together by overriding the default value and setting the reload interval to 0. This is extremely useful when trying to reduce overall cpu cycles consumed by an idle application server.

Class loading and update detection settings

JSP Reload Interval – This feature is similar to the application class reload interval, but extends to an application's JSP files.

JavaServer Pages (JSP) runtime reloading settings

Data Source Connection Pool Size – The minimum and maximum connection pool size should be increased to reduce time spent waiting to obtain a datasource connection from the connection pool. In this case, the maximum pool size is set to correspond to the default maximum Web Container and ORB thread pool size of 50. In clustered environments, this setting may overwhelm the database server and should be adjusted accordingly.

Connection pool settings

Data Source Prepared Statement Cache Size – The prepared statement cache optimizes the processing of prepared and callable statements by caching them in a least-recently-used (LRU) cache. Most applications will utilize more than 10 prepared statements. Consequently, the scripts for the performance test and production environments increase this value to 50. The Tivoli Performance Viewer

can be used to monitor LRU discards from the prepared statement cache to determine if the cache size should be increased.

Data access tuning parameters
WebSphere Application Server data source properties

ORB Pass-By-Reference – This feature allows the server to use pass-by-reference semantics instead of pass-by-value semantics for Enterprise JavaBean™ (EJB) invocations where the EJB client and remote target exist within the same JVM. This optimization essentially treats EJBs implementing a remote interface as local and avoids the requisite object copy. In some cases, this can improve performance by over 50%.

Object Request Broker service settings

Thread Pool Size – The default size of the various threads pools within the application server are generally well suited for Production and Test environments. However, for Development environments, the minimum and maximum sizes for the Web Container, ORB, Default thread pools can be reduced to reduce the memory footprint requirements of the server.

Thread pool settings

IBM, the IBM (logo), and WebSphere are trademarks or registered trademarks of International Business Machines Corporation in the United States, other countries, or both.

Solaris, 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, or service names may be trademarks or service marks of others.

References in this publication to IBM products or services do not imply that IBM intends to make them available in all countries in which IBM operates. The following paragraph does not apply to the United Kingdom or any other country where such provisions are inconsistent with local law:

INTERNATIONAL BUSINESS MACHINES CORPORATIONPROVIDES THIS PUBLICATION "AS IS" WITHOUT WARRANTYOF ANY KIND, EITHER EXPRESS OR IMPLIED, INCLUDING, BUTNOT LIMITED TO, THE IMPLIED WARRANTIES OF NON-INFRINGEMENT, MERCHANTABILITY OR FITNESS FOR APARTICULAR PURPOSE.

Some states do not allow disclaimer of express or implied warranties in certain transactions, therefore, this statement may not apply to you.

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

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

The information in this publication is provided AS IS without warranty. Such information was obtained from publicly available sources, is current as of January 2009, and is subject to change. Any performance data included in the paper was obtained in the specific operating environment and is provided as an illustration. Performance in other operating environments may vary. More specific information about the capabilities of products described should be obtained from the suppliers of those products.