



Sun Java™ System

Application Server 7 Performance Tuning Guide

2004Q2

Sun Microsystems, Inc.
4150 Network Circle
Santa Clara, CA 95054
U.S.A.

Part No: 817-5051

Copyright © 2004 Sun Microsystems, Inc., 4150 Network Circle, Santa Clara, California 95054, U.S.A. All rights reserved.

Sun Microsystems, Inc. has intellectual property rights relating to technology embodied in the product that is described in this document. In particular, and without limitation, these intellectual property rights may include one or more of the U.S. patents listed at <http://www.sun.com/patents> and one or more additional patents or pending patent applications in the U.S. and in other countries.

THIS PRODUCT CONTAINS CONFIDENTIAL INFORMATION AND TRADE SECRETS OF SUN MICROSYSTEMS, INC. USE, DISCLOSURE OR REPRODUCTION IS PROHIBITED WITHOUT THE PRIOR EXPRESS WRITTEN PERMISSION OF SUN MICROSYSTEMS, INC.

U.S. Government Rights - Commercial software. Government users are subject to the Sun Microsystems, Inc. standard license agreement and applicable provisions of the FAR and its supplements.

Use is subject to license terms. This distribution may include materials developed by third parties.

Sun, Sun Microsystems, the Sun logo, Java, Solaris, Sun™ ONE, Sun™ ONE Studio, iPlanet, J2EE, J2SE, Enterprise JavaBeans, EJB, JavaServer Pages, JSP, JDBC, JDK, JVM, Java Naming and Directory Interface, JavaMail, and the Java Coffee Cup logo are trademarks or registered trademarks of Sun Microsystems, Inc. in the U.S. and other countries.

All SPARC trademarks are used under license and are trademarks or registered trademarks of SPARC International, Inc. in the U.S. and other countries. Products bearing SPARC trademarks are based upon architecture developed by Sun Microsystems, Inc.

UNIX is a registered trademark in the U.S. and in other countries, exclusively licensed through X/Open Company, Ltd.

This product is covered and controlled by U.S. Export Control laws and may be subject to the export or import laws in other countries. Nuclear, missile, chemical biological weapons or nuclear maritime end uses or end users, whether direct or indirect, are strictly prohibited. Export or reexport to countries subject to U.S. embargo or to entities identified on U.S. export exclusion lists, including, but not limited to, the denied persons and specially designated nationals lists is strictly prohibited.

DOCUMENTATION IS PROVIDED "AS IS" AND ALL EXPRESS OR IMPLIED CONDITIONS, REPRESENTATIONS AND WARRANTIES, INCLUDING ANY IMPLIED WARRANTY OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE OR NON-INFRINGEMENT, ARE DISCLAIMED, EXCEPT TO THE EXTENT THAT SUCH DISCLAIMERS ARE HELD TO BE LEGALLY INVALID.

Copyright © 2004 Sun Microsystems, Inc., 4150 Network Circle, Santa Clara, California 95054, Etats-Unis. Tous droits réservés.

Sun Microsystems, Inc. détient les droits de propriété intellectuelle relatifs à la technologie incorporée dans le produit qui est décrit dans ce document. En particulier, et ce sans limitation, ces droits de propriété intellectuelle peuvent inclure un ou plus des brevets américains listés à l'adresse <http://www.sun.com/patents> et un ou les brevets supplémentaires ou les applications de brevet en attente aux Etats-Unis et dans les autres pays.

CE PRODUIT CONTIENT DES INFORMATIONS CONFIDENTIELLES ET DES SECRETS COMMERCIAUX DE SUN MICROSYSTEMS, INC. SON UTILISATION, SA DIVULGATION ET SA REPRODUCTION SONT INTERDITES SANS L'AUTORISATION EXPRESSE, ECRITE ET PREALABLE DE SUN MICROSYSTEMS, INC.

L'utilisation est soumise aux termes de la Licence. Cette distribution peut comprendre des composants développés par des tierces parties.

Sun, Sun Microsystems, le logo Sun, Java, Solaris, Sun™ ONE, Sun™ ONE Studio, iPlanet, J2EE, J2SE, Enterprise JavaBeans, EJB, JavaServer Pages, JSP, JDBC, JDK, JVM, Java Naming and Directory Interface, JavaMail, et le logo Java Coffee Cup sont des marques de fabrique ou des marques déposées de Sun Microsystems, Inc. aux Etats-Unis et dans d'autres pays.

Toutes les marques SPARC sont utilisées sous licence et sont des marques de fabrique ou des marques déposées de SPARC International, Inc. aux Etats-Unis et dans d'autres pays. Les produits portant les marques SPARC sont basés sur une architecture développée par Sun Microsystems, Inc.

UNIX est une marque déposée aux Etats-Unis et dans d'autres pays et licenciée exclusivement par X/Open Company, Ltd.

Ce produit est soumis à la législation américaine en matière de contrôle des exportations et peut être soumis à la réglementation en vigueur dans d'autres pays dans le domaine des exportations et importations. Les utilisations, ou utilisateurs finaux, pour des armes nucléaires, des missiles, des armes biologiques et chimiques ou du nucléaire maritime, directement ou indirectement, sont strictement interdites. Les exportations ou réexportations vers les pays sous embargo américain, ou vers des entités figurant sur les listes d'exclusion d'exportation américaines, y compris, mais de manière non exhaustive, la liste de personnes qui font objet d'un ordre de ne pas participer, d'une façon directe ou indirecte, aux exportations des produits ou des services qui sont régis par la législation américaine en matière de contrôle des exportations et la liste de ressortissants spécifiquement désignés, sont rigoureusement interdites.

LA DOCUMENTATION EST FOURNIE "EN L'ETAT" ET TOUTES AUTRES CONDITIONS, DECLARATIONS ET GARANTIES EXPRESSES OU TACITES SONT FORMELLEMENT EXCLUES, DANS LA MESURE AUTORISEE PAR LA LOI APPLICABLE, Y COMPRIS NOTAMMENT TOUTE GARANTIE IMPLICITE RELATIVE A LA QUALITE MARCHANDE, A L'APTITUDE A UNE UTILISATION PARTICULIERE OU A L'ABSENCE DE CONTREFAÇON.

Contents

About This Guide	7
Who Should Use This Guide	7
Using the Documentation	8
How This Guide Is Organized	10
Documentation Conventions	11
General Conventions	11
Conventions Referring to Directories	12
Contacting Sun	13
Give Us Feedback	13
Obtain Training	13
Contact Product Support	13
 Chapter 1 About Application Server Performance	 15
Process Overview	15
The Importance of Tuning	16
General Tuning Concepts	16
Performance Tuning Sequence	18
Configuration Files	18
Logging and Performance	19
 Chapter 2 Tuning Your Application	 21
Java Programming Guidelines	21
J2EE Programming Guidelines	24
Servlets and JSPs	24
EJBs	26
EJB Pool and Cache	28
Transactions	29

JDBC and Database Access	30
JMS	31
Chapter 3 Tuning the Application Server	33
HTTP Server Tuning	33
Enabling Statistics with stats-xml	34
Monitoring Current Activity Using the perfdump Utility	37
Using Statistics to Tune Your Server	40
Busy Functions	52
Using Performance Buckets	53
Configuring the File Cache	56
Tuning the ACL User Cache	60
Using Quality of Service	62
Threads, Processes, and Connections	62
Improving Java Performance	65
Miscellaneous init.conf Directives	66
Miscellaneous obj.conf Parameters	68
Scaling Your Server	69
Connection Pool Tuning	71
JDBC Connection Pool Tuning	73
Monitoring the JDBC Connection Pools	74
JSP and Servlet Tuning	75
Suggested Coding Practices for JSP's and Servlets	75
Configuration Settings that Affect JSP/Servlet Performance	76
Performance Tuning for EJBs	77
Monitoring EJBs and Containers	77
Tuning the EJB Container	79
Guide to Using Tunables	79
EJB Descriptor Properties	80
Tuning the EJB Pool	82
Tuning the EJB Cache	83
Performance Considerations for Various Types of EJBs	83
Related EJB Considerations	86
ORB Tuning	92
How a Client Connects to the ORB	93
Enabling the High performance CORBA Util Delegate	93
Monitoring the ORB	93
Tuning the ORB	94
Guide to using Tunables	97
Thread Pool Sizing	98
Related Considerations	99
Transaction Manager Tuning	100
Monitoring the Transaction Manager	100

Tuning the Transaction Manager	101
automatic-recovery	102
keypoint-interval	102
References	102
Chapter 4 Tuning the Java Runtime System	105
Using Alternate Threads	105
Managing Memory and Allocation	106
Tuning the Garbage Collector	106
Tracing Garbage Collection	108
Specifying Other Garbage Collector Settings	109
Tuning the Java Heap	110
HotSpot Virtual Machine Tuning Options	112
Chapter 5 Tuning the Operating System	115
Tuning Parameters	115
Solaris File Descriptor Setting	117
General Performance tuning for solaris on x86	118
Tuning for Linux platforms	119
Increase the number of file descriptors	120
Change the virtual memory settings	121
Ensure that the Network interface is operating in full duplex mode	121
Tune disk I/O performance	121
Tune the TCP/IP stack	122
Chapter 6 Tuning for High-Availability	125
Tuning HADB	125
Disk Usage	125
Memory Allocation	128
Performance	128
Operating System Configuration	135
Tuning the Application Server for High-Availability	137
Configuring and Tuning the Application Server	137
Appendix A Common Performance Problems	145
check-acl Server Application Functions	145
Low-Memory Situations	146
Under-Throttled Server	146
Cache Not Utilized	146
Keep-Alive Connections Flushed	147
Log File Modes	148
Additional Resources	148

Index149

About This Guide

The Sun Java™ System Application Server Standard Edition and Enterprise Edition provides a high-performance Java 2 Platform, Enterprise Edition (J2EE™)-compatible platform suitable for broad deployment of application services and web services.

This *Performance Tuning Guide* discusses the various subsystems, features, and tools inside the Application Server and how to tune them for maximum performance and reliability.

This preface addresses the following topics:

- [Who Should Use This Guide](#)
- [Using the Documentation](#)
- [How This Guide Is Organized](#)
- [Documentation Conventions](#)
- [Contacting Sun](#)

Who Should Use This Guide

This guide is intended for server administrators, J2EE developers, network administrators, and evaluators. This guide helps you tune the Application Server for maximum performance and reliability. It is recommended that you backup your configuration files, before changing the configuration settings on Application Server.

This guide assumes you are familiar with the following topics:

- Internet and World Wide Web

- Java programming
- J2EE application model
- Application Servers
- Your operating system:
 - Solaris™
 - Linux
 - Solaris on x86

Using the Documentation

The Sun Java System Application Server Standard and Enterprise Edition manuals are available as online files in Portable Document Format (PDF) and Hypertext Markup Language (HTML).

The following table lists tasks and concepts described in the Sun Java System Application Server manuals. The manuals marked *(updated for 7 2004Q2)* have been updated for the Sun Java System Application Server Standard and Enterprise Edition 7 2004Q2 release. The manuals not marked in this way have not been updated since the version 7 Enterprise Edition release.

Table 1 Sun Java System Application Server Documentation Roadmap

For information about	See the following
<i>(Updated for 7 2004Q2)</i> Late-breaking information about the software and the documentation. Includes a comprehensive, table-based summary of supported hardware, operating system, JDK, and JDBC/RDBMS.	<i>Release Notes</i>
Sun Java System Application Server 7 overview, including the features available with each product edition.	<i>Product Overview</i>
Diagrams and descriptions of server architecture and the benefits of the Sun Java System Application Server architectural approach.	<i>Server Architecture</i>
<i>(Updated for 7 2004Q2)</i> How to get started with the Sun Java System Application Server product. Includes a sample application tutorial. There are two guides, one for Standard Edition and one for Enterprise Edition.	<i>Getting Started Guide</i>
<i>(Updated for 7 2004Q2)</i> Installing the Sun Java System Application Server Standard Edition and Enterprise Edition software and its components, such as sample applications and the Administration interface. For the Enterprise Edition software, instructions are provided for implementing the high-availability configuration.	<i>Installation Guide</i>

Table 1 Sun Java System Application Server Documentation Roadmap (*Continued*)

For information about	See the following
(<i>Updated for 7 2004Q2</i>) Evaluating your system needs and enterprise to ensure that you deploy Sun Java System Application Server in a manner that best suits your site. General issues and concerns that you must be aware of when deploying an application server are also discussed.	<i>System Deployment Guide</i>
Creating and implementing Java™ 2 Platform, Enterprise Edition (J2EE™ platform) applications intended to run on the Sun Java System Application Server that follow the open Java standards model for J2EE components such as servlets, Enterprise JavaBeans™ (EJBs™), and JavaServer Pages™ (JSPs™). Includes general information about application design, developer tools, security, assembly, deployment, debugging, and creating lifecycle modules. A comprehensive Sun Java System Application Server glossary is included.	<i>Developer's Guide</i>
(<i>Updated for 7 2004Q2</i>) Creating and implementing J2EE web applications that follow the Java™ Servlet and JavaServer Pages (JSP) specifications on the Sun Java System Application Server. Discusses web application programming concepts and tasks, and provides sample code, implementation tips, and reference material. Topics include results caching, JSP precompilation, session management, security, deployment, SHTML, and CGI.	<i>Developer's Guide to Web Applications</i>
(<i>Updated for 7 2004Q2</i>) Creating and implementing J2EE applications that follow the open Java standards model for enterprise beans on the Sun Java System Application Server. Discusses Enterprise JavaBeans (EJB) programming concepts and tasks, and provides sample code, implementation tips, and reference material. Topics include container-managed persistence, read-only beans, and the XML and DTD files associated with enterprise beans.	<i>Developer's Guide to Enterprise JavaBeans Technology</i>
(<i>Updated for 7 2004Q2</i>) Creating Application Client Container (ACC) clients that access J2EE applications on the Sun Java System Application Server.	<i>Developer's Guide to Clients</i>
Creating web services in the Sun Java System Application Server environment.	<i>Developer's Guide to Web Services</i>
(<i>Updated for 7 2004Q2</i>) Java™ Database Connectivity (JDBC™), transaction, Java Naming and Directory Interface™ (JNDI), Java™ Message Service (JMS), and JavaMail™ APIs.	<i>Developer's Guide to J2EE Services and APIs</i>
Creating custom NSAPI plug-ins.	<i>Developer's Guide to NSAPI</i>
(<i>Updated for 7 2004Q2</i>) Information and instructions on the configuration, management, and deployment of the Sun Java System Application Server subsystems and components, from both the Administration interface and the command-line interface. Topics include cluster management, the high-availability database, load balancing, and session persistence. A comprehensive Sun Java System Application Server glossary is included.	<i>Administration Guide</i>
(<i>Updated for 7 2004Q2</i>) Editing Sun Java System Application Server configuration files, such as the <code>server.xml</code> file.	<i>Administrator's Configuration File Reference</i>

Table 1 Sun Java System Application Server Documentation Roadmap (Continued)

For information about	See the following
Configuring and administering security for the Sun Java System Application Server operational environment. Includes information on general security, certificates, and SSL/TLS encryption. HTTP server-based security is also addressed.	<i>Administrator's Guide to Security</i>
Configuring and administering service provider implementation for J2EE™ Connector Architecture (CA) connectors for the Sun Java System Application Server. Topics include the Administration Tool, Pooling Monitor, deploying a JCA connector, and sample connectors and sample applications.	<i>J2EE CA Service Provider Implementation Administrator's Guide</i>
(Updated for 7 2004Q2) Migrating your applications to the new Sun Java System Application Server programming model, specifically from iPlanet Application Server 6.x and Sun ONE Application Server 7.0. Includes a sample migration.	<i>Migrating and Redeploying Server Applications Guide</i>
(Updated for 7 2004Q2) How and why to tune your Sun Java System Application Server to improve performance.	<i>Performance Tuning Guide</i>
(Updated for 7 2004Q2) Information on solving Sun Java System Application Server problems.	<i>Troubleshooting Guide</i>
(Updated for 7 2004Q2) Information on solving Sun Java System Application Server error messages.	<i>Error Message Reference</i>
(Updated for 7 2004Q2) Utility commands available with the Sun Java System Application Server; written in manpage style.	<i>Utility Reference Manual</i>
Using the Sun™ Java System Message Queue 3.5 software.	The Sun Java System Message Queue documentation at: http://docs.sun.com/db?p=prod/s1.s1msgqu

How This Guide Is Organized

This guide is organized as follows:

- Chapter 1, “About Application Server Performance”** describes the techniques and processes involved in tuning Application Server.
- Chapter 2, “Tuning Your Application”** describes practices and configuration settings you can use with your applications to ensure maximum performance.
- Chapter 3, “Tuning the Application Server”** describes how you can configure the application server for your needs.
- Chapter 4, “Tuning the Java Runtime System”** describes how you can configure the Java Virtual Machine to work optimally with the Application Server.

[Chapter 5, “Tuning the Operating System”](#) describes how you can configure your operating system to work optimally with the Application Server.

[Chapter 6, “Tuning for High-Availability”](#) describes how the high-availability database (HADB) that is used for storing persistent session state needs to be tuned for the Application Server. It also discusses how you can configure the high availability features of Application Server for your application.

[Appendix A, “Common Performance Problems”](#) describes common performance problems users face when the Application Server is used as a classic web server.

An [“Index”](#) is provided for quick reference lookups to key performance terms.

Documentation Conventions

This section describes the types of conventions used throughout this guide:

- [General Conventions](#)
- [Conventions Referring to Directories](#)

General Conventions

The following general conventions are used in this guide:

- **File and directory paths** are given in UNIX® format (with forward slashes separating directory names). For Windows versions, the directory paths are the same, except that backslashes are used to separate directories.
- **URLs** are given in the format:

`http://server.domain/path/file.html`

In these URLs, *server* is the server name where applications are run; *domain* is your Internet domain name; *path* is the server’s directory structure; and *file* is an individual filename. Italic items in URLs are placeholders.

- **Font conventions** include:
 - The monospace font is used for sample code and code listings, API and language elements (such as function names and class names), file names, pathnames, directory names, and HTML tags.
 - *Italic* type is used for code variables.

- *Italic* type is also used for book titles, emphasis, variables and placeholders, and words used in the literal sense.
- **Bold** type is used as either a paragraph lead-in or to indicate words used in the literal sense.
- **Installation root directories** for most platforms are indicated by *install_dir* in this document. Exceptions are noted in [“Conventions Referring to Directories” on page 12](#).

By default, the location of *install_dir* on **most** platforms is:

- Solaris and Linux file-based installations:

user's home directory/sun/appserver7

- Windows, all installations:

system drive: \Sun\AppServer7

For the platforms listed above, *default_config_dir* and *install_config_dir* are identical to *install_dir*. See [“Conventions Referring to Directories” on page 12](#) for exceptions and additional information.

- **Instance root directories** are indicated by *instance_dir* in this document, which is an abbreviation for the following:
default_config_dir/domains/*domain*/*instance*
- **UNIX-specific descriptions** throughout this manual apply to the Linux operating system as well, except where Linux is specifically mentioned.

Conventions Referring to Directories

By default, when using the Solaris package-based or Linux RPM-based installation, the application server files are spread across several root directories. This guide uses the following document conventions to correspond to the various default installation directories provided:

- *install_dir* refers to */opt/SUNWappserver7*, which contains the static portion of the installation image. All utilities, executables, and libraries that make up the application server reside in this location.
- *default_config_dir* refers to */var/opt/SUNWappserver7/domains*, which is the default location for any domains that are created.

- *install_config_dir* refers to `/etc/opt/SUNWappserver7/config`, which contains installation-wide configuration information such as licenses and the master list of administrative domains configured for this installation.

Contacting Sun

You might want to contact Sun Microsystems in order to:

- [Give Us Feedback](#)
- [Obtain Training](#)
- [Contact Product Support](#)

Give Us Feedback

If you have general feedback on the product or documentation, please send this to <http://www.sun.com/hwdocs/feedback>

Obtain Training

Application Server training courses are available at:

http://training.sun.com/US/catalog/enterprise/web_application.html/

Visit this site often for new course availability on the Sun Java System Application Server.

Contact Product Support

If you have problems with your system, contact customer support using one of the following mechanisms:

- The online support web site at:
<http://www.sun.com/supporttraining/>
- The telephone dispatch number associated with your maintenance contract

Please have the following information available prior to contacting support. This helps to ensure that our support staff can best assist you in resolving problems:

- Description of the problem, including the situation where the problem occurs and its impact on your operation
- Machine type, operating system version, and product version, including any patches and other software that might be affecting the problem. Here are some of the commonly used commands:
 - **Solaris:** `pkginfo, showrev`
 - **Linux:** `rpm`
 - **All:** `asadmin version --verbose`
- Detailed steps on the methods you have used to reproduce the problem
- Any error logs or core dumps
- Configuration files such as:
 - *instance_dir*/config/server.xml
 - a web application's `web.xml` file, when a web application is involved in the problem
- For an application, whether the problem appears when it is running in a cluster or standalone

About Application Server Performance

This chapter discusses the following topics:

- [Process Overview](#)
- [The Importance of Tuning](#)
- [General Tuning Concepts](#)
- [Performance Tuning Sequence](#)
- [Configuration Files](#)
- [Logging and Performance](#)

Process Overview

The following table outlines the overall administration process, and shows where Performance Tuning fits in the sequence. .

Table 1-1 Performance Tuning Roadmap

Step	Description of Task	Location of Instructions
1	Design: Decide on your high-availability topology and set up your systems	<i>System Deployment Guide</i>
2	Capacity Planning: Make sure the systems have sufficient resources to perform well.	<i>System Deployment Guide, Appendix X</i>
3	Installation: Install the HADB software with or without the Application Server software	<i>Installation Guide</i>

Table 1-1 Performance Tuning Roadmap

Step	Description of Task	Location of Instructions
4	Deployment: Run your applications. Familiarize yourself with how to configure and administer the Application Server subsystems and components.	<i>Administrator's Guide</i>
5.	Tuning: Tune your application, Java Runtime System, operating system, high-availability database (HADB), and the Application Server.	<i>Performance Tuning Guide</i>

The Importance of Tuning

Performance can be significantly enhanced by adjusting a few deployment descriptor settings or server configuration file modifications. However, it is important to understand the environment and performance goals. An optimal configuration for a production environment may not necessarily be optimal for a development environment. This guide helps you to understand the tuning and sizing options available, providing you the capabilities and practices to obtain the best performance out of your Application Server.

General Tuning Concepts

The following table describes factors that affect performance. The left most column describes the general concept, the second column gives the practical ramifications of the concept, the third column describes the measurements, and the right most column describes the value sources.

Table 1-2 Factors That Affect Performance - Applying Concepts

Concept	Applying the Concept	Measurement	Value Sources
User Load	Concurrent Sessions at Peak Load	Transactions Per Minute (TPM) Web Interactions Per Second (WIPS)	((Number of Concurrent Users at Peak Load) * Expected Response Time) / (Time between clicks) For example, (100 Concurrent Users * 2 seconds Response Time) / (10 seconds between clicks) = 20.

Table 1-2 Factors That Affect Performance - Applying Concepts

Concept	Applying the Concept	Measurement	Value Sources
Application Scalability	Transaction Rate measured on one CPU	TPM or WIPS	Measured from workload benchmark. Needs to be performed at each tier.
	vertical scalability (additional performance for additional CPU)	Percentage gain per additional CPU	Based on curve fitting from benchmark. Perform tests while gradually increasing the number of CPUs. Identify the “knee” of the curve, where additional CPUs are providing uneconomical gains in performance. Requires tuning as described in later chapters of this guide. Needs to be performed at each tier and iterated if necessary. Stop here if this meets performance requirements.
	horizontal scalability (additional performance for additional server)	Percentage gain per additional server process and/or hardware node.	Use a well tuned single application server instance, as in previous step. Measure how much each additional server instance and/or hardware node improves performance.
Safety Margins	High Availability Requirements.	If system should cope with failures, size the system to meet performance requirements assuming that one or more application server instances are non functional	Different equations used if High Availability is required.
	Slack for unexpected peaks	It is desirable to operate a server at less than its benchmarked peak, for some safety margin	80% system capacity utilization at peak loads, may work for most installations. Measure your deployment under real and simulated peak loads.

Performance Tuning Sequence

Ideally, tuning is performed in the following sequence:

- [Tuning Your Application](#)
- [Tuning the Application Server](#)
- [Tuning HADB](#)
- [Tuning the Java Runtime System](#)
- [Tuning the Operating System](#)

Application tuning is listed first because it tends to produce unexpectedly large improvements in performance—larger than any other tuning operation. So, ideally, that is the first step in the tuning cycle.

The remaining steps are performed by the system administrator. You take those steps when the application has already been tuned, or when application tuning has to wait and you want to improve performance as much as possible in the meantime.

Configuration Files

The files `init.conf`, `<instance_name>-obj.conf`, and `server.xml` are Application Server configuration files containing many attributes that can be modified to improve performance. They are frequently mentioned in this guide and can be found in the following directory:

```
<APPSERVER_HOME>/domains/<DOMAIN_NAME>/<SERVER_NAME>/config/
```

The following configuration files are located in the directory:

- `admch`
- `admsh`
- `backup`
- `certmap.conf`
- `dbswitch.conf`
- `default-web.xml`
- `generated.server1.acl`

- `genwork.server1.acl`
- `init.conf`
- `keyfile`
- `login.conf`
- `mime.types`
- `obj.conf`
- `secure.seed`
- `server.policy`
- `server.xml`
- `server1-obj.conf`
- `sun-acc.xml`

`APPSERVER_HOME` is the installation directory for the Application Server. `DOMAIN_NAME` and `SERVER_NAME` refer to the domain and server names for the server instance to be configured.

The `config/backup` directories contain a replica of the server configuration files. These files are created by the administration server instance. In general, users should not change these files. If the `config` files are edited, make a copy of the files and place them in the `backup` directory. Additionally, the server instance should be restarted.

Logging and Performance

The Application Server produces log messages and exception stack trace output that gets written to the log file. These log messages and exception stacks can be found in the `logs` directory of the instance. Naturally, the volume of log activity can impact server performance; particularly in benchmarking situations.

By default, the log level is set to `INFO`. The log level can be set for all the server subsystems by changing the attribute level in the `log_service` element. You can override the logging level by adjusting it at a particular subsystem. For example, `mdb_container` can produce log messages at a different level than server default by adjusting the `log_level` attribute under the `mdb_container` element. To get more debug messages, set the log level to `FINE`, `FINER`, or `FINEST`. Under benchmarking conditions, it may be appropriate to set the log level to `SEVERE`.

NOTE	When using the load balancer, the load balancer plug-in uses the web server logging mechanism to write log messages. The load balancer plug-in uses the default logging level on Sun ONE Web Server (<code>INFO</code>), and Apache Web Server (<code>WARN</code>). The application server log levels - <code>FINE</code> , <code>FINER</code> , and <code>FINEST</code> map to the <code>DEBUG</code> level on the web server.
-------------	-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------

Tuning Your Application

The following sections provide a comprehensive guide to tuning your applications for maximum performance. A complete guide to writing high performance Java and J2EE applications is beyond the scope of this document.

The following topics are discussed in this chapter:

- [Java Programming Guidelines](#)
- [J2EE Programming Guidelines](#)

Java Programming Guidelines

This section covers issues related to Java coding and performance. The guidelines outlined are not specific to Application Server, but are general rules that are useful in many situations. For a complete discussion of Java coding best practices, refer to the Java BluePrints at <http://java.sun.com/blueprints/performance/index.html>.

Avoid serialization and deserialization, if possible

In Java, serialization and deserialization of objects is a CPU-intensive procedure and is likely to slow down your application. Use the `transient` keyword to reduce the amount of data serialized. Customized `readObject()` and `writeObject()` methods may be beneficial, in some cases.

Use `StringBuffer.append()` instead of `"+"`

In Java, Strings are immutable – they never change after creation. In the following sequence:

```
String str = "testing";  
str = str + "abc";
```

is understood by the compiler as:

```
String str = "testing";
StringBuffer tmp = new StringBuffer(str);
tmp.append("abc");
str = tmp.toString();
```

Therefore, copying is inherently expensive and can become a significant factor in hindering performance in case it is overused. Instead, the use of `StringBuffer.append()` is recommended.

Assign null to variables you're finished with

Explicitly assigning a null value to variables that are no longer needed helps the garbage collector to easily identify the parts of memory that can be safely reclaimed. While Java automates memory management, it does not prevent usage of excessive amounts of memory or memory leaks.

An application may induce memory leaks by holding on to objects without releasing references. This prevents the Java garbage collector from reclaiming those objects, and results in increasing amounts of memory being used. Explicitly nullifying references to unnecessary variables, after each transaction, allows the garbage collector to reclaim memory.

One way to detect memory leaks is to employ profiling tools and take memory snapshots after each transaction. A memory leak free application, in steady state, will show a steady active heap memory, after garbage collections.

Only declare methods as final when absolutely necessary

Modern optimizing dynamic compilers can perform inlining and other inter-procedural optimizations, even if Java methods are not `final`. Use the keyword as it was originally intended i.e., for program architecture and maintainability reasons. If you are absolutely certain that a method must not be overridden, use the `final` keyword.

Declare constants as static final

The dynamic compiler can perform some constant folding optimizations easily, when the hint is provided.

Avoid finalizers

Adding finalizers to your code makes the garbage collector more expensive and unpredictable. The virtual machine does not guarantee the time at which finalizers are run. Finalizers may not always be executed, before the program exits. Releasing critical resources in `finalize()` methods may lead to unpredictable application behavior.

Declare method arguments final

Declare method arguments `final` if they are not modified in the method. In general, declare all variables `final` if they are not modified after being initialized or set to some value.

Synchronize only when necessary

Do not synchronize code blocks or methods unless synchronization is required. Keep synchronized blocks or methods as short as possible to avoid scalability bottlenecks. Use the Java Collections Framework for unsynchronized data structures instead of more expensive alternatives such as `java.util.Hashtable`.

Use DataHandlers for SOAP attachments

Using a `javax.activation.DataHandler` for a SOAP attachment to improve performance.

JAXRPC specifies:

- A mapping of certain MIME types to Java types.
- That any MIME type is mappable to a `javax.activation.DataHandler`.

As a result, you can send an attachment (`.gif` or XML document) as a SOAP attachment to an RPC style webservice by utilizing the Java type mappings. When passing in any of the mandated Java type mappings (appropriate for your attachment mime type) as an argument for your webservice, JAXRPC runtime will handle these as SOAP attachments for you.

For example, to send out an `image/gif` attachment, you could use `java.awt.Image`, or you could create a `DataHandler` wrapper over your image. The advantages of using the wrapper are:

- Reduced coding

You can reuse "generic attachment code" to handle the attachments, because the `DataHandler` determines the content type of the contained data automatically. This feature is especially useful when using a document style service. Since the content is known at runtime, there is no need to make calls to `attachment.setContent(stringContent, "image/gif")`, for example.

- Improved Performance

In informal tests, utilizing the `DataHandler` wrappers has shown throughput twice as high vs. use of `image/gif` MIME types, and approximately 1.5 times the throughput when using these wrappers for `text/xml` or `java.awt.Image` for `image/*` types.

For more information, consult the *Application Server Developer's Guide to Web Services*.

J2EE Programming Guidelines

The J2EE model defines a framework for enterprise application development. It defines containers for the basic software components (JSPs, servlets and EJBs) and container services (JAAS, JDBC, JNDI, and JTA for example). While all parts of the J2EE model have their uses, the following sections discuss issues to keep in mind while designing the application architecture.

Servlets and JSPs

Many applications running on the Application Server are serviced by JSPs or servlets in the presentation tier. Servlets and JSPs are entry points to EJBs, on which more complex transactional business logic is implemented. It is not uncommon to see moderately complex transactional business applications coded entirely using Servlets and other J2EE APIs.

Avoid shared, modified class variables

In the case of the servlet multithread model which is the default model, a single instance of a servlet is created for each application server instance. All requests for a servlet on that application instance share the same servlet instance. This can lead to thread contention if there are synchronization blocks in the servlet code. So avoid using shared modified class variables, since they create the need for synchronization.

Create sessions sparingly, invalidate whenever possible

Session creation is not for free. If a session is not required do not create one. Invalidate sessions when they are no longer needed.

Use `<%page session="false"%>`

Use the directive `<%page session="false"%>` to prevent HTTP Sessions from being automatically created when they are not necessary in JSPs.

Avoid large object graphs in an `HttpSession`

Do not store large object graphs inside an `HttpSession`. They force Java serialization and add computational overhead.

Don't cache transaction data in `HttpSession`

Note that `HttpSession` access is not transactional. Do not use it as a cache of transactional data, which is typically kept in the database and accessed using Entity beans. Transactions will rollback, upon failures, to their original state. However, stale and inaccurate data may remain in `HttpSession` objects. Application Server provides “read-only” bean managed persistence entity beans for more cached access to read only data.

Set the reap interval value carefully

Modifying the reap interval setting (`reapIntervalSeconds` in `server.xml`) can improve performance, but setting it without considering the nature of your sessions and business logic can cause data inconsistency.

For example, if you set the reap interval parameter to 60 seconds, the value of session data will be recorded every 60 seconds. But if your servlet's client accesses the servlet to update a counter (for example, bidding price) at 20 second increments, then inconsistencies will result.

Here's a possible scenario:

- Bidding starts at \$5, in 60 seconds the value recorded will be \$8 (+ 3 "20 second intervals").
- During the next 40 seconds, the client starts incrementing the price. The value the client sees is \$10.
- During the client's 20 second rest, the Application Server stops and starts in 10 seconds. As a result, the latest value recorded at the 60 second interval (\$8) is be loaded into the session.

- The Client clicks again expecting to see \$11; but instead sees is \$9, which is incorrect.

So, to avoid data inconsistencies, take into the account the expected behavior of the application when adjusting the reap interval.

Keep session size small

If possible, keep the session size small, so you incur less latency in response times. If possible, you should keep the session size below 7 Kb.

Checkpoint only when needed

In High Availability mode, when using Stateful session beans, consider checkpointing only those methods that alter the state of the bean significantly. This will reduce the number of times the bean state has to be checkpointed into the persistent store.

EJBs

The following guidelines can improve performance of EJB components in J2EE Applications. For more information, see the *Application Server Developer's Guide to Enterprise Java Beans Technology*.

NOTE	Keep in mind that decomposing an application into a moderate to large number of separate EJBs can create an application performance degradation and more overhead. EJBs, unlike JavaBeans, are not simply Java objects. EJBs are higher level entities than Java objects. They are components with remote call interface semantics, security semantics, transaction semantics, and properties.
-------------	------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------

Cache EJB references in the servlet

To avoid a JNDI lookup for every request, cache EJB references in the servlet.

Cache EJBHomes in the servlet's init() method

Repeated lookups to a frequently used home interface can be expensive.

Use `setSessionContext()` or `ejbCreate()` to cache bean resources

This is again an example of using bean lifecycle methods to perform application actions, only once, where possible. Do not forget to place code to release acquired resources in the `ejbRemove()` method.

Minimize the database transaction isolation level

Reduce the database transaction isolation level when appropriate. Reduced isolation levels reduce work in the database tier, and could lead to better application performance. However, this must be done after carefully analyzing the database table usage patterns. The Application Server allows database isolation level to be set under `<jdbc-connection-pool>` in server configuration file. For more details on server configuration see the *Application Server Administrators Guide* and the *Application Server Administrator's Configuration File Reference*.

Use `rmic without -nolocalstubs` when co-hosting

The Application Server ORB provides a mechanism for optimizing calls made from clients which are co-hosted in the same Java Virtual Machine as the server—for example, when a servlet calls an Enterprise Java Bean or one Enterprise Java Bean calling another enterprise bean located on the same server instance. When the servlets and EJB's are running in the same Java virtual machine, run the `rmic` compiler *without* the `-nolocalstubs` flag. This is the default setting i.e `-nolocalstubs` does not appear in the server configuration file.

Use `rmic with -nolocalstubs` when running remotely

If the application architecture is such that EJBs are hosted on a remote Application Server, the default behavior must be changed. This can be done via the administration command-line interface `asadmin`, on the server instance where the application is going to be deployed. The `rmic` options appear under the JVM Settings tab of the browser based administration interface. Use of local stubs gives significant performance enhancements and is the default behavior of the stub generator.

Use pass by reference, when possible

If you are sure about the access paths to the underlying data, then the beans can be configured to use *pass by reference*. This avoids argument copying on method invocations and result copying on method return. However, problems will arise if the data is modified by another source, during the invocation. This value can be set in the `sun-ejb-jar.xml` deployment descriptor as follows:

```
<pass-by-reference>true</pass-by-reference>
```

on a per EJB granularity.

Remove Stateful session beans when they are no longer needed.

Removing them avoids passivation of the Stateful Session beans, and the attendant disk I/O operations.

Prefer high performance beans

Here are the EJB types are listed below, from the highest performance to the lowest:

- Stateless Session Beans and Message Driven Beans
- Stateful Session Beans
- Entity Beans, with bean managed persistence, configured as read only
- Entity Beans with Container Managed Persistence (CMP)
- Entity Beans with Bean Managed Persistence (BMP)

EJB Pool and Cache

Both *stateless* session beans and *entity* beans can be pooled to improve server performance. In addition, both *stateful* session beans and *entity* beans can be cached to improve performance.

Table 2-1 Bean Type Pooling or Caching

Bean Type	Pooled	Cached
Stateless Session	Yes	No
Stateful Session	No	Yes
Entity	Yes	Yes

The difference between a pooled bean and a cached bean is that pooled beans are all equivalent and indistinguishable from one another. Cached beans, on the contrary, contain conversational state in the case of Stateful Session beans, and are associated with a primary key in the case of Entity beans. Entity beans are removed from the pool and added to the cache on `ejbActivate()` and removed from the cache and added to the pool on `ejbPassivate()`. `ejbActivate()` is called by the container when a needed Entity bean is not in the cache. `ejbPassivate()` is called by the container when the cache grows beyond its configured limits.

Here are some steps you can take when tuning the EJB pool and cache settings:

Tune the pool size

The EJB pool is used by Stateless Session and Entity EJB's. Keeping in mind how you use Stateless Session EJB's and the amount of traffic your server handles, tune the pool size to prevent excessive creation and deletion. Refer to the *bean-pool* `sun-ejb-jar.xml` deployment descriptor.

Tune the EJB cache size and time-out settings

The Cache is used by Stateful Session EJB's. Keeping in mind how your applications uses Stateful Session EJB's and the amount of traffic your server handles, tune the EJB cache size and time-out settings to minimize the number of activations and passivations. Refer to the *bean-cache* element in the `sun-ejb-jar.xml` deployment descriptor.

Explicitly call `remove()`

Allow Stateful Session EJB's to be removed from the container cache by explicitly calling of the `remove()` method in the client.

Tune the entity EJB's pool size

Entity Beans use both the EJB pool and cache settings. Tune the entity EJB's pool size to minimize the creation and destruction of beans. Prepopulating the pool with a non-zero steady size is useful to get better response for initial requests.

Cache and release bean specific resources

Use the `setEntityContext()` method to cache bean specific resources and release them using the `unsetEntityContext()` method.

Use Lazy Loading

Use Lazy Loading to avoid unnecessary preloading of child data.

Identify read-only beans

Configure read-only entity beans for read only operations.

Transactions

Here are some steps you can take when using transactions.

Use container managed transactions

Container managed transactions are preferred for consistency, and provide better performance.

Don't encompass user input time

To avoid resources being held unnecessarily for long periods, a transaction should not encompass user input or user think time.

Identify non-transactional methods

Declare non-transactional methods of session EJB's with 'NotSupported' or 'Never' transaction attributes. These attributes can be found in the `ejb-jar.xml` deployment descriptor file. Transactions should span the minimum time possible since they lock database rows.

Use TX_REQUIRED for long transaction chains

For very large transaction chains, use the transaction attribute `TX_REQUIRED`. To ensure EJB methods in a call chain, use the same transaction.

Use lowest cost database locking

Use the lowest cost locking available from the database that is consistent with any transaction. Commit the data after the transaction completes rather than after each method call.

Use XA capable data sources only when needed

When multiple database resources, connector resources and/or JMS resources are involved in one transaction, a distributed or global transaction needs to be performed. This requires XA capable resource managers and data sources. Use XA capable data sources, only when two or more data source are going to be involved in a transaction. If a database participates in some distributed transactions, but mostly in local or single database transactions, it is advisable to register two `<jdbc-resource>` elements in server configuration file and use the appropriate resource in the application.

JDBC and Database Access

Here are some steps you can take when tuning the JDBC Connection Pool:

For large amounts of data, use JDBC directly

When dealing with large amounts of data, such as searching a large database, use JDBC directly rather than using Entity EJB's.

Encapsulate business logic in Entity EJBs

Combine business logic with the Entity EJB that holds the data needed for that logic to process.

Close connections after use

To ensure that connections are returned to the pool, always close the connections after use.

JMS

Here are some steps you can take when using JMS:

Tune the Message-Driven EJB's pool size

Tune the Message-Driven EJB's pool size to optimize the concurrent processing of messages.

Cache and release bean specific resources

Use the `setMessageDrivenContext()` or `ejbCreate()` method to cache bean specific resources, and release those resources from the `ejbRemove()` method.

Limit Usage of JMS connections

When designing an application that utilizes JMS connections make sure you use a methodology that sparingly uses connections, by either pooling them or using the same connection for multiple sessions.

There is a limitation in iMQ, in that it doesn't pool connections, so the application developer has to work around that limitation (which is not keeping in the spirit of the Application Server will provide all resources).

The JMS connection uses 2 threads and the sessions use 1 thread each. Since these threads are not taken from a pool and the resultant objects aren't pooled, you could run out of memory during periods of heavy usage.

One workaround is to move `createTopicConnection` into the `init` of the `jsp`.

NOTE Make sure you specifically close you session, or it will stay open, which will tie up resources.

References

- For details on performance guidelines regarding Java, see <http://java.sun.com/blueprints/performance/index.html>
- For details on optimizing EJB's, see <http://developer.java.sun.com/developer/technicalArticles/ebeans/sevenrules/>
- For tips on maximizing the Web Server's performance, see the Sun ONE Web Server 6.1 Performance Tuning, Sizing, and Scaling Guide at <http://docs.sun.com/db/doc/817-1836-10>

Tuning the Application Server

This chapter describes some ways to tune the Application Server for optimum performance. It is separated into the following sections:

- [HTTP Server Tuning](#)
- [Connection Pool Tuning](#)
- [JSP and Servlet Tuning](#)
- [Performance Tuning for EJBs](#)
- [Performance Considerations for Various Types of EJBs](#)
- [ORB Tuning](#)
- [Related Considerations](#)
- [Monitoring the Transaction Manager](#)

HTTP Server Tuning

Monitoring and tuning the HTTP server instances that handle client requests are important parts of ensuring peak Application Server performance. This section covers the following topics related to HTTP Server Tuning:

- [Enabling Statistics with stats-xml](#)
- [Monitoring Current Activity Using the perfdump Utility](#)
- [Using Statistics to Tune Your Server](#)
- [Busy Functions](#)
- [Using Performance Buckets](#)

- [Configuring the File Cache](#)
- [Tuning the ACL User Cache](#)
- [Using Quality of Service](#)
- [Threads, Processes, and Connections](#)
- [Improving Java Performance](#)
- [Miscellaneous init.conf Directives](#)
- [Miscellaneous obj.conf Parameters](#)
- [Scaling Your Server](#)

Enabling Statistics with stats-xml

Users must enable statistics with `stats-xml` when they wish to use existing monitoring tools like `perfdump` or create similar customized tools.

To enable the statistics using `stats-xml`, follow these steps:

1. Under the default object in `<serverInstance>-obj.conf`, add the following line:

```
NameTrans fn="assign-name" from="/stats-xml/*" name="stats-xml"
```

Note:

Both this entry and the “perf” entry must appear before the document-root function. Otherwise, they will be ignored. The resulting file should look something like this:

```
NameTrans fn=assign-name from="/stats-xml/*" name="stats-xml"
NameTrans fn=assign-name from="/.perf" name="perf"
...
NameTrans fn=document-root root="$docroot"
```

2. Add the following Service function to `<serverInstance>-obj.conf`:

```
<Object name="stats-xml">
Service fn="stats-xml"
</Object>
```

The following figure shows a sample `<serverInstance>-obj.conf` which has `stats-init xml` enabled.

```

<Object name="default">
AuthTrans fn="match-browser" browser="*MSIE*" ssl-unclean-shutdown="true"
NameTrans fn="ntrans-j2ee" name="j2ee"
NameTrans fn=pfx2dir from=/mc-icons dir="/export/home/software/ias70_gold/lib/ic
ons" name="es-internal"
NameTrans fn=document-root root="$docroot"
# Line added below for Enabling stats-xml
NameTrans fn="assign-name" from="/stats-xml/*" name="stats-xml"
PathCheck fn=unix-uri-clean
PathCheck fn="check-acl" acl="default"
PathCheck fn=find-pathinfo
PathCheck fn=find-index index-names="index.html,home.html"
ObjectType fn=type-by-extension
ObjectType fn=force-type type=text/plain
Service method=(GET|HEAD) type=magnus-internal/imagemap fn=imagemap
Service method=(GET|HEAD) type=magnus-internal/directory fn=index-common
Service method=(GET|HEAD|POST) type="*~magnus-internal/*" fn=send-file
Error fn="error-j2ee"
AddLog fn=flex-log name="access"
</Object>

<Object name="j2ee">
ObjectType fn=force-type type=text/html
Service fn="service-j2ee" method="*"
</Object>

<Object name="cgi">
ObjectType fn=force-type type=magnus-internal/cgi
Service fn=send-cgi user="$user" group="$group" chroot="$chroot" dir="$dir" nice
="$nice"
</Object>

<Object name="es-internal">
PathCheck fn="check-acl" acl="es-internal"
</Object>

# Service Function Added for enabling stats-xml
<Object name="stats-xml">
Service fn="stats-xml"
</Object>
~

```

Figure 3-1 statsxml-obj in <serverInstance>-obj.conf

3. Add the stats-init Server Application Function (SAF) to <serverInstance>-obj.conf. Here's an example:

```
Init fn="stats-init" update-interval="5" virtual-servers="2000"
profiling="yes"
```

The following figure shows a sample init.conf file which has stats-init Server Application Function (SAF) included.

```

xterm
NetsiteRoot /export/home/software/ias70_gold
ServerID sizing
ServerName iasperf
PidLog /export/home/software/ias70_gold/domains/domain1/sizing/logs/pid
User vm95884
DNS off
Security off
RqThrottle 128
StackSize 131072
TempDir /tmp/sizing-8f612f65

Init fn=flex-init access="$accesslog" format.access="%Ses->client.ip% - %Req->va
rs.auth-user% [%SYSDATE%] \"%Req->reqpb.clf-request%\" %Req->srvhdrs.clf-status%
%Req->srvhdrs.content-length%"
Init fn=stats-init
Init fn="load-modules" shlib="/export/home/software/ias70_gold/lib/libj2eeplugin
.so" funcs="init-j2ee,ntrans-j2ee,service-j2ee,error-j2ee" shlib_flags="(global|
now)"
Init fn="init-j2ee" LateInit=yes
#Line Added for enabling stats-xml
Init fn="stats-init" update-interval="5" virtual-servers="2000" profiling="yes"

```

Figure 3-2 Enabling Statistics with stats-xml

The above example shows you can also designate the following:

- **update-interval.** The period in seconds between statistics updates. A higher setting (less frequent) will be better for performance. The minimum value is 1; the default value is 5.
- **virtual-servers.** The maximum number of virtual servers for which you track statistics. This number should be set equal to or higher than the number of virtual servers configured. Smaller numbers result in lower memory usage. The minimum value is 1; the default is 1000.
- **profiling.** Enable NSAPI performance profiling. The default is "no" which results in slightly better server performance.

For more information on editing the configuration files, see the *Application Server NSAPI Programmer's Guide*.

Monitoring Current Activity Using the perfdump Utility

The `perfdump` utility is an SAF built into Application Server. It collects various pieces of performance data from the Application Server internal statistics and displays them in ASCII text. The `perfdump` utility allows you to monitor a greater variety of statistics.

Installing the perfdump Utility

The following figure provides a sample of the `<serverInstance>-obj.conf` file with the `perfdump` utility configured.



```

xterm
<Object name="default">
AuthTrans fn="match-browser" browser="*MSIE*" ssl-unclean-shutdown="true"
# Line Added for enabling perf dump
NameTrans fn=assign-name from="/.perf" name="perf"
NameTrans fn=ntrans-j2ee name="j2ee"
NameTrans fn=pfx2dir from="/mc-icons dir="/export/home/software/ias70_gold/lib/ico
ns" name="es-internal"
NameTrans fn=document-root root="$docroot"
# Line added below for Enabling stats-xm1
NameTrans fn="assign-name" from="/stats-xm1/*" name="stats-xm1"
PathCheck fn=unix-uri-clean
PathCheck fn="check-acl" acl="default"
PathCheck fn=find-pathinfo
PathCheck fn=find-index index-names="index.html,home.html"
ObjectType fn=type-by-extension
ObjectType fn=force-type type=text/plain
Service method=(GET|HEAD) type=magnus-internal/imagemap fn=imagemap
Service method=(GET|HEAD) type=magnus-internal/directory fn=index-common
Service method=(GET|HEAD|POST) type="magnus-internal/*" fn=send-file
Error fn="error-j2ee"
AddLog fn=flex-log name="access"
</Object>

<Object name="j2ee">
ObjectType fn=force-type type=text/html
Service fn="service-j2ee" method="*"
</Object>

<Object name="cgi">
ObjectType fn=force-type type=magnus-internal/cgi
Service fn=send-cgi user="$user" group="$group" chroot="$chroot" dir="$dir" nice
="$nice"
</Object>

<Object name="es-internal">
PathCheck fn="check-acl" acl="es-internal"
</Object>

# Service Function Added for enabling stats-xm1
<Object name="stats-xm1">
Service fn="stats-xm1"
</Object>

# Service Function Added for enabling perfdump
<Object name="perf">
Service fn="service-dump"
</Object>

```

Figure 3-3 Sample `<instance-name>-obj.conf` file with `perfdump` Configured

To install `perfdump`, you need to make the following modifications in `<serverInstance>-obj.conf` file:

1. Add the following object to your `<serverInstance>-obj.conf` file after the default object:

```
<Object name="perf">  
Service fn="service-dump"  
</Object>
```

2. Add the following to the default object:

```
NameTrans fn=assign-name from="/.perf" name="perf"
```

3. If not already enabled, enable `stats-xml`.

If you need to enable `stats-xml`, see [“Enabling Statistics with stats-xml.”](#)

4. Restart your server software.

5. Access `perfdump` by entering this URL:

```
http://yourhost/.perf
```

6. You can request the `perfdump` statistics and specify how frequently (in seconds) the browser should automatically refresh. This example sets the refresh to every 5 seconds:

```
http://yourhost/.perf?refresh=5
```

The following figure shows a sample `perfdump` output.

```

appservd pid: 4223

Sun ONE Application Server 7.0 B10/08/2002 14:05 (SunOS DOMESTIC)

Server started Thu Nov 07 16:50:23 2002
Process 4223 started Thu Nov 07 16:50:23 2002

ConnectionQueue:
-----
Current/Peak/Limit Queue Length 0/1/4096
Total Connections Queued 1
Average Queueing Delay 0.09 milliseconds

ListenSocket http-listener-1:
-----
Address http://0.0.0.0:1025
Acceptor Threads 1
Default Virtual Server sizing

KeepAliveInfo:
-----
KeepAliveCount 0/256
KeepAliveHits 0
KeepAliveFlushes 0
KeepAliveRefusals 0
KeepAliveTimeouts 1
KeepAliveTimeout 30 seconds

SessionCreationInfo:
-----
Active Sessions 1
Total Sessions Created 48/128

CacheInfo:
-----
enabled yes
CacheEntries 3/1024
Hit Ratio 3/6 ( 50.00%)
Maximum Age 30

Native pools:
-----
NativePool:
Idle/Peak/Limit 1/1/128
Work Queue Length/Peak/Limit 0/0/0

Server DNS cache disabled

Async DNS disabled

Performance Counters:
-----
Average Total Percent
Total number of requests: 1
Request processing time: 0.0036 0.0036

default-bucket (Default bucket)
Number of Requests: 1 (100.00%)
Number of Invocations: 20 (100.00%)
Latency: 0.0004 0.0004 ( 10.56%)
Function Processing Time: 0.0032 0.0032 ( 83.44%)
Total Response Time: 0.0036 0.0036 (100.00%)

Sessions:
-----
Process Status Function
4223 response service-dump

```

Figure 3-4 Sample perfdump Output

For more information on editing the configuration files, see the *Application Server Developer's Guide to NSAPI*.

Using Statistics to Tune Your Server

This section describes the information available through the `perfdump` utility and discusses how to tune some parameters to improve your server's performance. The default tuning parameters are appropriate for all sites except those with very high volume. The only parameters that large sites may regularly need to change are `RqThrottle`, `MaxKeepAliveConnections`, and `KeepAliveTimeout`, which can be tuned by using the web-based Administration interface or by directly editing the `<serverInstance>-obj.conf` file.

The `perfdump` utility monitors statistics in the following categories:

- [Connection Queue Information](#)
- [HTTP Listener Information](#)
- [Keep-Alive/Persistent Connection Information](#)
- [Session Creation Information](#)
- [Cache Information](#)
- [Thread Pools](#)
- [DNS Cache Information](#)

Connection Queue Information

Connection queue information shows the number of sessions in the queue, and the average delay before the connection is accepted.

Following is an example of how these statistics are displayed in `perfdump`:

```
ConnectionQueue:
-----
Current/peak/limit queue length 0/48/5000
Total connections queued 3753
Average queueing delay 0.0013 seconds
```

Current /peak /limit

Current/peak/limit queue length shows, in order:

- The number of connections currently in the queue.
- The largest number of connections that have been in the queue simultaneously.

- The maximum size of the connection queue.

Tuning

If the peak queue length is close to the limit, you may wish to increase the maximum connection queue size to avoid dropping connections under heavy load.

You can increase the connection queue size by:

- Setting or changing the value of `ConnQueueSize` from the web-based Administration interface. Follow the path `http-server-Advanced-performance`.
- Editing the `ConnQueueSize` directive in `init.conf`.

CAUTION Setting the connection queue too high can degrade server performance. It was designed to prevent the server from becoming overloaded with connections it cannot handle. If your server is overloaded and you increase the connection queue size, the latency of request handling will increase further, and the connection queue will fill up again.

Total Connections Queued

Total connections queued is the total number of times a connection has been queued. This includes newly accepted connections and connections from the keep-alive system.

This is a gathered statistic and is not tunable.

Average Queuing Delay

Average queueing delay is the average amount of time a connection spends in the connection queue. This represents the delay between when a request connection is accepted by the server, and a request processing thread (also known as a session) begins servicing the request.

This is a gathered statistic and is not tunable.

HTTP Listener Information

The HTTP listener information includes the IP address, port number, number of acceptor threads, and the default virtual server for the HTTP listener. For tuning purposes, the most important field in the HTTP listener information is the number of acceptor threads.

You can have many listeners enabled for virtual servers, but you will at least have one (usually `http://0.0.0.0:80`) enabled for your default server instance.

```

Http listeners1:
-----

Address http://0.0.0.0:1890

Acceptor threads 1

Default virtual server test

```

Tuning

You can create and configure HTTP listeners through the web-based Administration interface. For more information, see *Application Server Administrator's Guide*.

If you have created multiple HTTP Listeners, `perfdump` displays them all.

Set the TCP/IP listen queue size for all HTTP Listeners by:

- Editing the `ListenQ` parameter in `init.conf`.
- Entering the value in the `Listen Queue Size` field of the Performance Tuning page of the web-based Administration interface.

Address

This field contains the base address that this listener is listening on. It contains the IP address and the port number.

If your HTTP listener listens on all IP addresses for the machine, the IP part of the address is 0.0.0.0.

Tuning

This setting is tunable when you edit a listen socket. If you specify an IP address other than 0.0.0.0, the server will make one less system call per connection. Specify an IP address other than 0.0.0.0 for the best possible performance.

Acceptor Threads

Acceptor threads are threads that wait for connections. The threads accept connections and put them in a queue where they are then picked up by worker threads. Ideally, you want to have enough acceptor threads so that there is always one available when a user needs one, but few enough so that they do not put too much of a burden on the system. A good rule is to have one acceptor thread per CPU on your system. You can increase this value to about double the number of CPUs if you find indications of TCP/IP listen queue overruns.

Tuning

You can tune the number of acceptor threads by selecting the HTTP listener node and changing the number of acceptor threads on the right hand side of the Advanced category.

Default Virtual Server

Software virtual servers work using the HTTP 1.1 Host header. If the end user's browser does not send the host header, or if the server cannot find the virtual server specified by the Host header, Application Server handles the request using a default virtual server. Also, for hardware virtual servers, if the application server cannot find the virtual server corresponding to the IP address, it displays the default virtual server. You can configure the default virtual server to send an error message or serve pages from a special document root.

Tuning

You can specify a default virtual server for an individual listen socket and the server instance. If a given HTTP listener does not have a default virtual server, the server instance's default virtual server is used.

You can specify a default virtual server for a listen socket using the web-based Administration interface. You can set or change the default virtual server information using the Edit HTTP Listener page on the Preferences Tab of the web-based Administration interface for the HTTP server. The settings for the default virtual server are on the Connection Group Settings page that appears when you click Groups.

Keep-Alive/Persistent Connection Information

This section provides statistics about the server's HTTP-level keep-alive system.

The following example shows the keep-alive statistics displayed by `perfdump`:

```
KeepAliveInfo:
-----
KeepAliveCount 1/256
KeepAliveHits 4
KeepAliveFlushes 1
KeepAliveTimeout 30 seconds
```

NOTE The name "keep-alive" should not be confused with TCP "keep-alives." Also, note that the name "keep-alive" was changed to "Persistent Connections" in HTTP/1.1, but the `.perf` continues to refer to them as "KeepAlive" connections.

Both HTTP 1.0 and HTTP 1.1 support the ability to send multiple requests across a single HTTP session. A web server can receive hundreds of new HTTP requests per second. If every request was allowed to keep the connection open indefinitely, the server could become overloaded with connections. On Unix systems, this could easily lead to a file table overflow.

To deal with this problem, the server maintains a "Maximum number of 'waiting' keep-alive connections" counter. A 'waiting' keep-alive connection has fully completed processing the previous request, and is waiting for a new request to arrive on the same connection. If the server has more than the maximum waiting connections open when a new connection waits for a keep-alive request, the server closes the oldest connection. This algorithm keeps an upper bound on the number of open waiting keep-alive connections that the server can maintain.

The Application Server does not always honor a keep-alive request from a client. The following conditions cause the server to close a connection even if the client has requested a keep-alive connection:

- `KeepAliveTimeout` is set to 0.
- `MaxKeepAliveConnections` count is exceeded.
- Dynamic content, such as a CGI, does not have an HTTP `content-length` header set. This applies only to HTTP 1.0 requests. If the request is HTTP 1.1, the server honors keep-alive requests even if the `content-length` is not set. The server can use chunked encoding for these requests if the client can handle them (indicated by the request header `transfer-encoding:chunked`). For more information regarding chunked encoding, see the *Application Server Developer's Guide to NSAPI*.
- Request is not HTTP GET or HEAD.
- The request was determined to be bad. For example, if the client sends only headers with no content.

KeepAliveThreads

You can configure the number of threads used in the keep-alive system by:

- Editing the `KeepAliveThreads` parameter in `init.conf`.

- Setting or changing the `KeepAliveThreads` value in the web-based Administration interface. Follow the path: HTTP Server, Advanced Tab, Keep-Alive sub-menu.

KeepAliveCount

This setting has two numbers:

- Number of connections in keep-alive mode
- Maximum number of connections allowed in keep-alive mode simultaneously

Tuning

You can tune the maximum number of sessions the server allows to wait at one time before closing the oldest connection by:

- Editing the `MaxKeepAliveConnections` parameter in the `init.conf` file.
- Setting or changing the `MaxKeepAliveConnections` value in the web-based Administration interface.

NOTE The number of connections specified by `MaxKeepAliveConnections` is divided equally among the keep-alive threads. If `MaxKeepAliveConnections` is not equally divisible by `KeepAliveThreads`, the server may allow slightly more than `MaxKeepAliveConnections` simultaneous keep-alive connections.

KeepAliveHits

The number of times a request was successfully received from a connection that had been kept alive.

This setting is not tunable.

KeepAliveFlushes

The number of times the server had to close a connection because the `KeepAliveCount` exceeded the `MaxKeepAliveConnections`.

This setting is not tunable.

KeepAliveTimeout

Specifies the number of seconds the server allows a client connection to remain open with no activity. A web client may keep a connection to the server open so that multiple requests to one server can be serviced by a single network connection. Since a given server can handle a finite number of open connections, a high number of open connections will prevent new clients from connecting.

Tuning

You can change `KeepAliveTimeout` by:

- Editing the `KeepAliveTimeout` parameter in `init.conf`
- Setting or changing the `KeepAliveTimeout` value in the web-based Administration interface
- Entering the value in the HTTP Persistent Connection Timeout field of the Performance Tuning page in the web-based Administration interface.

KeepAliveQueryMeanTime

This parameter determines the intervals between the polling of Connections being handled by the `KeepAlive` Subsystem. If this parameter is set to a value `N` milliseconds, the Response time seen by a client which has requested persistent connections will have an overhead ranging from 0 to `N` milliseconds. This value is set by default to 1 millisecond (if unspecified in `init.conf`). This value works well if you expect a concurrent load of no more than about 300 `KeepAlive` connections. The default value can severely affect the scalability with higher concurrent loads. It is suggested that this be appropriately increased for higher connection load.

Tuning

You can change `KeepAliveQueryMeanTime` by editing the `KeepAliveQueryMeanTime` parameter in `init.conf`.

UseNativePoll

For Unix users, this parameter should be enabled for maximum performance.

To enable native poll for your keep-alive system from the web-based Administration interface, follow these steps:

1. Select the HTTPServer node for the server instance that needs to have this option turned on.
2. Select the Advanced Tab on the right hand side of the pane.
3. Select the Keep alive page tab.

4. Use the drop-down list to set `UseNativePoll` to ON.
5. Click OK.
6. Select the server instance tab from the tree view.
7. Select Apply Changes.
8. Restart the instance to cause your changes to take effect.

Session Creation Information

Session creation statistics are only displayed in `perfdump`. Following is an example of `SessionCreationInfo` displayed in `perfdump`:

```
SessionCreationInfo:
-----

Active Sessions 1

Total Sessions Created 48/512
```

Active Sessions shows the number of sessions (request processing threads) currently servicing requests.

Total Sessions Created shows both the number of sessions that have been created and the maximum number of sessions allowed.

Reaching the maximum number of configured threads is not necessarily undesirable. It is not necessary to automatically increase the number of threads in the server. Reaching this limit means that the server needed this many threads at peak load. As long as it was able to serve requests in a timely manner, the server is adequately tuned. However, at this point connections will queue up in the connection queue, potentially overflowing it. If you check your `perfdump` output on a regular basis and notice that total sessions created is often near the `RqThrottle` maximum, you should consider increasing your thread limits.

Tuning

You can increase your thread limits by:

- Editing the `RqThrottle` parameter in `init.conf`.
- Setting or changing the `RqThrottle` value via the web-based Administration interface.
- Entering the value in the Maximum Simultaneous Requests field of the Performance Tuning page in the web-based Administration interface.

Cache Information

The Cache information section provides statistics on how your file cache is being used. The file cache caches static content so that the server handles requests for static content quickly.

Following is an example of how the cache statistics are displayed in `perfdump`:

```
CacheInfo:
-----

enabled yes

CacheEntries 5/1024

Hit Ratio 93/190 ( 48.95%)

Maximum age 30
```

enabled

If the cache is disabled, the rest of this section is not displayed.

Tuning

The cache is enabled by default. You can disable it by:

- Unselecting it from the File Cache Configuration page under File Caching Tab in the web-based Administration interface for the HTTP Server instance.
- Editing the `FileCacheEnable` parameter in the `nsfc.conf` file. For more information, see the *Application Server Developer's Guide to NSAPI*.

CacheEntries

The number of current cache entries and the maximum number of cache entries are both displayed. A single cache entry represents a single URI.

Tuning

You can set the maximum number of cached entries by:

- Entering a value in the Maximum Number of Files field on the File Cache Configuration page under Preferences in the Server Manger.
- Creating or editing the `MaxFiles` parameter in the `nsfc.conf` file. For more information, see the *Application Server Developer's Guide to NSAPI*.

Hit Ratio (CacheHits / CacheLookups)

The hit ratio gives you the number of file cache hits versus cache lookups. Numbers approaching 100% indicate the file cache is operating effectively, while numbers approaching 0% could indicate that the file cache is not serving many requests.

This setting is not tunable.

Maximum Age

The maximum age displays the maximum age of a valid cache entry. This parameter controls how long cached information is used after a file has been cached. An entry older than the maximum age is replaced by a new entry for the same file.

Tuning

If your web site's content changes infrequently, you may want to increase this value for improved performance. You can set the maximum age by:

- Entering or changing the value in the Maximum Age field of the File Cache Configuration page in the web-based Admin Console for the HTTP server node and selecting the File Caching Tab.
- Editing the `MaxAge` parameter in the `nsfc.conf` file. For more information, see the *Application Server Developer's Guide to NSAPI*.

Thread Pools

The thread pools can be configured through the web-based Administration interface:

- Thread Pools (Unix)

Thread Pools (Unix)

Since threads on Unix are always operating system (OS)-scheduled, as opposed to user-scheduled, Unix users do not need to use native thread pools. Therefore, this option is not offered in a Unix user interface. However, you can edit the OS-scheduled thread pools and add new thread pools, if needed, using the web-based Administration interface.

Idle/Peak/Limit

Idle indicates the number of threads that are currently idle. Peak indicates the peak number in the pool. Limit indicates the maximum number of native threads allowed in the thread pool, and is determined by the setting of

`NativePoolMaxThreads`.

Tuning

You can modify the `NativePoolMaxThreads` by:

- Editing the `NativePoolMaxThreads` parameter in `init.conf`
- Entering or changing the value in the Maximum Threads field of the Native Thread Pool page in the Performance tab for the Http server node via the administration interface

Work Queue Length /Peak /Limit

These numbers refer to a queue of server requests that are waiting for the use of a native thread from the pool. The Work Queue Length is the current number of requests waiting for a native thread.

Peak is the highest number of requests that were ever queued up simultaneously for the use of a native thread since the server was started. This value can be viewed as the maximum concurrency for requests requiring a native thread.

Limit is the maximum number of requests that can be queued at one time to wait for a native thread, and is determined by the setting of `NativePoolQueueSize`.

Tuning

You can modify the `NativePoolQueueSize` by:

- Editing the `NativePoolQueueSize` parameter in `init.conf`
- Entering or changing the value in the Queue Size field of the Native Thread Pool page in the Performance tab for the HTTP server node via the web-based Administration interface.

DNS Cache Information

The DNS cache caches IP addresses and DNS names. Your server's DNS cache is disabled by default. In the DNS Statistics for Process ID All page under Monitor in the web-based Administration interface the following statistics are displayed:

enabled

If the DNS cache is disabled, the rest of this section is not displayed.

Tuning

By default, the DNS cache is off. You can enable DNS caching by:

- Adding the following line to `init.conf`:

```
Init fn=dns-cache-init
```

- Setting the DNS value to “Perform DNS lookups on clients accessing the server” via the web-based Administration interface.

CacheEntries (CurrentCacheEntries / MaxCacheEntries)

The number of current cache entries and the maximum number of cache entries. A single cache entry represents a single IP address or DNS name lookup. The cache should be as large as the maximum number of clients that will access your web site concurrently. Note that setting the cache size too high will waste memory and degrade performance.

Tuning

You can set the maximum size of the DNS cache by:

- Adding the following line to the `init.conf` file:

```
Init fn=dns-cache-init cache-size=1024
```

The default cache size is 1024

- Entering or changing the value in the Size of DNS Cache field of the Performance Tuning page in the web-based Administration interface.

HitRatio (CacheHits / CacheLookups)

The hit ratio displays the number of cache hits versus the number of cache lookups.

This setting is not tunable.

NOTE	If you turn off DNS lookups on your server, host name restrictions will not work and hostnames will not appear in your log files. Instead, you'll see IP addresses.
-------------	---------------------------------------------------------------------------------------------------------------------------------------------------------------------

Caching DNS Entries

You can also specify whether to cache the DNS entries. If you enable the DNS cache, the server can store hostname information after receiving it. If the server needs information about the client in the future, the information is cached and available without further querying. You can specify the size of the DNS cache and an expiration time for DNS cache entries. The DNS cache can contain 32 to 32768 entries; the default value is 1024. Values for the time it takes for a cache entry to expire can range from 1 second to 1 year specified in seconds; the default value is 1200 seconds (20 minutes).

Limit DNS Lookups to Asynchronous

It is recommended that you do not use DNS lookups in server processes because they are so resource-intensive. If you must include DNS lookups, be sure to make them asynchronous.

enabled

If asynchronous DNS is disabled, the rest of this section will not be displayed.

Tuning

You can enable asynchronous DNS by:

- Adding `AsyncDNS ON` in the `init.conf` file.
- Setting the `AsyncDNS` value to ON in the web-based Administration interface.
- Selecting Async DNS Enabled from the Performance Tuning page under Preferences in the Server Manger.

NameLookups

The number of name lookups (DNS name to IP address) that have been done since the server was started.

This setting is not tunable.

AddrLookups

The number of address loops (IP address to DNS name) that have been done since the server was started.

This setting is not tunable.

LookupsInProgress

The current number of lookups in progress.

This setting is not tunable.

Busy Functions

The default busy function returns a "503 Service Unavailable" response and logs a message if `LogVerbose` is enabled. You may wish to modify this behavior for your application. To better troubleshoot performance problems, it is useful to specify your own busy functions for any NSAPI function in the `<serverInstance>-obj.conf` file by including a service function in the configuration file in this format:

```
busy="<my-busy-function>"
```

For example, you could use this sample service function:

```
Service fn="send-cgi" busy="service-toobusy"
```

This allows different responses if the server becomes too busy in the course of processing a request that includes a number of types (such as `Service`, `AddLog`, and `PathCheck`). Note that your busy function will apply to all functions that require a native thread to execute when the default thread type is non-native.

To use your own busy function instead of the default busy function for the entire server, you can write an NSAPI `init` function that includes a `func_insert` call as shown below:

```
extern "C" NSAPI_PUBLIC int my_custom_busy_function(pblock *pb, Session
*sn, Request *rq);

my_init(pblock *pb, Session *, Request *)
{
    func_insert("service-toobusy", my_custom_busy_function);
}
```

Busy functions are never executed on a pool thread, so you must be careful to avoid using function calls that could cause the thread to block.

Using Performance Buckets

Performance buckets allow you to define buckets, and link them to various server functions. Every time one of these functions is invoked, the server collects statistical data and adds it to the bucket. For example, `send-cgi` and `NSServletService` are functions used to serve the CGI and Java servlet requests respectively. You can either define two buckets to maintain separate counters for CGI and servlet requests, or create one bucket that counts requests for both types of dynamic content. The cost of collecting this information is little and impact on the server performance is usually negligible. The following information is stored in a bucket:

- **Name of the bucket.** This name is used for associating the bucket with a function.
- **Description.** A description of the functions associated with the bucket.
- **Number of requests for this function.** The total number of requests that caused this function to be called.
- **Number of times the function was invoked.** This number may not coincide with the number of requests for the function because some functions may be executed more than once for a single request.

- **Function latency or the dispatch time.** The time taken by the server to invoke the function.
- **Function time.** The time spent in the function itself.

The default-bucket is pre-defined by the server. It records statistics for the functions not associated with any user defined bucket.

Configuration

You must specify all the configuration information for performance buckets in the `init.conf` and `<serverInstance>-obj.conf` files. Only the default bucket is automatically enabled.

The following examples show how to define new buckets in `init.conf`:

```
Init fn="define-perf-bucket" name="acl-bucket" description="ACL bucket"
```

```
Init fn="define-perf-bucket" name="file-bucket" description="Non-cached responses"
```

```
Init fn="define-perf-bucket" name="cgi-bucket" description="CGI Stats"
```

The prior example creates three buckets: `acl-bucket`, `file-bucket`, and `cgi-bucket`. To associate these buckets with functions, add `bucket=bucket-name` to the `<serverInstance>-obj.conf` function for which you wish to measure performance. For example:

```
PathCheck fn="check-acl" acl="default" bucket="acl-bucket"
...
Service method="(GET|HEAD|POST)" type="*~magnus-internal/*"
fn="send-file" bucket="file-bucket"
...
<Object name="cgi">
  ObjectType fn="force-type" type="magnus-internal/cgi"
  Service fn="send-cgi" bucket="cgi-bucket"
</Object>
```

Performance Report

The performance buckets information is located in the last section of the report that `perfdump` returns.

For more information, see [“Enabling Statistics with stats-xml”](#) and [“Using Performance Buckets.”](#)

The report contains the following information:

- Average, Total, and Percent columns give data for each requested statistic.

- Request Processing Time is the total time required by the server to process all the requests it has received so far.
- Number of Requests is the total number of requests for the function.
- Number of Invocations is the total number of times that the function was invoked. This differs from the number of requests in that a function can be called multiple times while processing one request. The percentage column for this row is calculated in reference to the total number of invocations for all the buckets.
- Latency is the time, in seconds, that the Application Server takes to prepare for calling the function.
- Function Processing Time is the time, in seconds, that the Application Server spent inside the function. The percentage of Function Processing Time and Total Response Time is calculated with reference to the total Request processing time.
- Total Response Time is the sum, in seconds, of Function Processing Time and Latency.

The following is an example of the performance bucket information available through `perfdump`:

Performance Counters:

Average Total Percent

Total number of requests: 474851

Request processing time: 0.0010 485.3198

Default Bucket (default-bucket)

Number of Requests: 597 (0.13%)

Number of Invocations: 9554 (1.97%)

Latency: 0.0000 0.1526 (0.03%)

Function Processing Time: 0.0256 245.0459 (50.49%)

Total Response Time: 0.0257 245.1985 (50.52%)

Configuring the File Cache

The Application Server uses a file cache to serve static information faster. The file cache contains information about files and static file content. The file cache also caches information that is used to speed up processing of server-parsed HTML.

The file cache is turned on by default. The file cache settings are contained in a file called `nsfc.conf`. You can use the web-based Administration interface to change the file cache settings.

To configure the file cache:

1. Select the File Caching tab of the HTTP server.
2. Check Enable File Cache, if not already selected.
3. Choose whether or not to transmit files.

When you enable Transmit File, the server caches open file descriptors for files in the file cache, rather than the file contents. `PR_TransmitFile` is used to send the file contents to a client. When Transmit File is enabled, the distinction normally made by the file cache between small, medium, and large files no longer applies since only the open file descriptor is being cached. By default, Transmit File is enabled on Windows, and disabled on UNIX. On UNIX, only enable Transmit File for platforms that have native OS support for `PR_TransmitFile`, like HP-UX and AIX. Don't enable it for other UNIX/Linux platforms.

4. Enter a size for the hash table.

The default size is twice the maximum number of files plus 1. For example, if your maximum number of files is set to 1024, the default hash table size is 2049.

5. Enter a maximum age in seconds for a valid cache entry.

By default, this is set to 30.

This setting controls how long cached information will continue to be used once a file has been cached. An entry older than `MaxAge` is replaced by a new entry for the same file, if the same file is referenced through the cache.

Set the maximum age based on whether the content is updated (existing files are modified) on a regular schedule or not. For example, if content is updated four times a day at regular intervals, you could set the maximum age to 21600 seconds (6 hours). Otherwise, consider setting the maximum age to the longest time you are willing to serve the previous version of a content file after the file has been modified.

6. Enter the Maximum Number of Files to be cached.

By default, this is set to 1024.

7. (Unix only) Enter medium and small file size limits in bytes.

By default, the Medium File Size Limit is set to 525000 (525 KB).

By default, Small File Size Limit is set to 2048.

The cache treats small, medium, and large files differently. The contents of medium files are cached by mapping the file into virtual memory (Unix platforms). The contents of "small" files are cached by allocating heap space and reading the file into it. The contents of "large" files (larger than "medium") are not cached, although information about large files is cached.

The advantage of distinguishing between small files and medium files is to avoid wasting part of many pages of virtual memory when there are lots of small files. So the Small File Size Limit is typically a slightly lower value than the VM page size.

8. (Unix only) Set the medium and small file space.

The medium file space is the size in bytes of the virtual memory used to map all medium sized files. By default, this is set to 10000000 (10MB).

The small file space is the size of heap space in bytes used for the cache, including heap space used to cache small files. By default, this is set to 1MB for Unix.

9. Click OK.

10. Click Apply.

11. Select Apply Changes to restart your server.

Using the nocache Parameter

You can use the parameter `nocache` for the Service function `send-file` to specify that files in a certain directory not be cached. For example, if you have a set of files that changes too rapidly for caching to be useful, you can put them in a directory and instruct the server not to cache files in that directory by editing the

`<serverInstance>-obj.conf` file.

For example:

```
<Object name=default>
...

```

```

NameTrans fn="pfx2dir" from="/myurl" dir="/export/mydir" name="myname"
...
Service method=(GET|HEAD|POST) type=~magnus-internal/* fn=send-file
...
</Object>

<Object name="myname">
Service method=(GET|HEAD) type=~magnus-internal/* fn=send-file
nocache=" "
</Object>

```

In the above example, the server does not cache static files from `/export/mydir/` when requested by the URL prefix `/myurl`.

File Cache Dynamic Control and Monitoring

You can add an object to `<serverInstance>-obj.conf` to dynamically monitor and control the `nsfc.conf` file cache while the server is running. To do this:

Add a `NameTrans` directive to the default object:

```
NameTrans fn="assign-name" from="/nsfc" name="nsfc"
```

Add an `nsfc` object definition:

```

<Object name="nsfc">
Service fn=service-nsfc-dump
</Object>

```

This enables the file cache control and monitoring function (`nsfc-dump`) to be accessed via the URI, `/nsfc.` By changing the `"from"` parameter in the `NameTrans` directive, a different URI can be used.

The following is an example of the information you receive when you access the URI:

```
Application Server File Cache Status (pid 7960)
```

```
The file cache is enabled.
```

```
Cache resource utilization
```

```
Number of cached file entries = 1039 (112 bytes each, 116368 total
bytes)
```

```
Heap space used for cache = 237641/1204228 bytes
```

```

Mapped memory used for medium file contents = 5742797/10485760 bytes
Number of cache lookup hits = 435877/720427 ( 60.50 %)
Number of hits/misses on cached file info = 212125/128556
Number of hits/misses on cached file content = 19426/502284
Number of outdated cache entries deleted = 0
Number of cache entry replacements = 127405
Total number of cache entries deleted = 127407
Number of busy deleted cache entries = 17

```

Parameter settings

```

HitOrder: false
CacheFileInfo: true
CacheFileContent: true
TransmitFile: false
MaxAge: 30 seconds
MaxFiles: 1024 files
SmallFileSizeLimit: 2048 bytes
MediumFileSizeLimit: 537600 bytes
CopyFiles: false
Directory for temporary files: /tmp/netscape/https-axilla.mcom.com
Hash table size: 2049 buckets

```

You can include a query string when you access the `/nsfc` URI. The following values are recognized:

- `?list` - Lists the files in the cache.
- `?refresh=n` - Causes the client to reload the page every `n` seconds.
- `?restart` - Causes the cache to be shut down and then restarted.
- `?start` - Starts the cache.
- `?stop` - Shuts down the cache.

If you choose the `?list` option, the file listing includes the file name, a set of flags, the current number of references to the cache entry, the size of the file, and an internal file ID value. The flags are as follows:

- **C** - File contents are cached.
- **D** - Cache entry is marked for delete.
- **E** - `PR_GetFileInfo()` returned an error for this file.
- **I** - File information (size, modify date, etc.) is cached.
- **M** - File contents are mapped into virtual memory.
- **O** - File descriptor is cached (when `TransmitFile` is set to true).
- **P** - File has associated private data (should appear on `shtml` files).
- **T** - Cache entry has a temporary file.
- **W** - Cache entry is locked for write access.

For sites with scheduled updates to content, consider shutting down the cache while the content is being updated, and starting it again after the update is complete. Although performance will slow down, the server operates normally when the cache is off.

Tuning the ACL User Cache

The ACL user cache is ON by default. Because of the default size of the cache (200 entries), the ACL user cache can be a bottleneck, or can simply not serve its purpose on a site with heavy traffic. On a busy site, more than 200 users can hit ACL-protected resources in less time than the lifetime of the cache entries. When this situation occurs, the Application Server has to query the LDAP server more often to validate users, which impacts performance.

This bottleneck can be avoided by increasing the size of the ACL cache with the `ACLUserCacheSize` directive in `init.conf`. Note that increasing the cache size will use more resources; the larger you make the cache the more RAM you'll need to hold it.

There can also be a potential (but much harder to hit) bottleneck with the number of groups stored in a cache entry (by default four). If a user belongs to five groups and hits five ACLs that check for these different groups within the ACL cache lifetime, an additional cache entry is created to hold the additional group entry. When there are two cache entries, the entry with the original group information is ignored.

While it would be extremely unusual to hit this possible performance problem, the number of groups cached in a single ACL cache entry can be tuned with the `ACLGroupCacheSize` directive.

ACL User Cache Directives

To adjust the ACL user cache values you will need to manually add the following directives to your `init.conf` file:

- `ACLCacheLifetime`
- `ACLUserCacheSize`
- `ACLGroupCacheSize`

ACLCacheLifetime

Set this directive to a number that determines the number of seconds before the cache entries expire. Each time an entry in the cache is referenced, its age is calculated and checked against `ACLCacheLifetime`. The entry is not used if its age is greater than or equal to the `ACLCacheLifetime`. The default value is 120 seconds. If this value is set to 0, the cache is turned off. If you use a large number for this value, you may need to restart the Application Server when you make changes to the LDAP entries. For example, if this value is set to 120 seconds, the Application Server might be out of sync with the LDAP server for as long as two minutes. If your LDAP is not likely to change often, use a large number.

ACLUserCacheSize

Set this directive to a number that determines the size of the User Cache (default is 200).

ACLGroupCacheSize

Set this directive to a number that determines how many group IDs can be cached for a single UID/cache entry (default is 4).

Verifying ACL User Cache Settings

With `LogVerbose` you can verify that the ACL user cache settings are being used. When `LogVerbose` is running, you should expect to see these messages in your error log when the server starts:

```
User authentication cache entries expire in ### seconds.
User authentication cache holds ### users.
Up to ### groups are cached for each cached user.
```

Tuning

You can turn `LogVerbose` ON by editing the `LogVerbose` parameter in `init.conf`.

CAUTION Do not turn on `LogVerbose` on a production server. Doing so degrades performance and increases the size of your error logs.

Using Quality of Service

The quality of service features let you limit the amount of bandwidth and number of connections for a server instance, class of virtual servers, or individual virtual server. You can set these performance limits, track them, and optionally enforce them.

For more information, see “Using Quality of Service” in the *Application Server Administrator's Guide*.

Threads, Processes, and Connections

In the Application Server, acceptor threads on an HTTP Listener accept connections and put them onto a connection queue. Session threads then pick up connections from the queue and service the requests. More session threads are posted if required at the end of the request. The policy for adding new threads is based on the connection queue state:

- Each time a new connection is returned, the number of connections waiting in the queue (the backlog of connections) is compared to the number of session threads already created. If it is greater than the number of threads, more threads are scheduled to be added the next time a request completes.
- The previous backlog is tracked, so that another `ThreadIncrement` number of threads are scheduled to be added if the following are true:
 - The number of threads are seen to be increasing over time.
 - The increase is greater than the `ThreadIncrement` value.
 - The number of session threads minus the backlog is less than the `ThreadIncrement` value.
- The process of adding new session threads is strictly limited by the `RqThrottle` value.

- To avoid creating too many threads when the backlog increases suddenly (such as the startup of benchmark loads), the decision whether more threads are needed is made only once every 16 or 32 times a connection is made based on how many session threads already exist.

The following directives affect the number and timeout of threads, processes, and connections can be tuned in the web-based Admin Console or `init.conf`:

- `ConnQueueSize`
- `HeaderBufferSize`
- `AcceptTimeOut`
- `KeepAliveThreads`
- `KeepAliveTimeout`
- `KernelThreads`
- `ListenQ`
- `MaxKeepAliveConnections`
- `MaxProcs` (Unix Only)
- `PostThreadsEarly`
- `RcvBufSize`
- `RqThrottle`
- `RqThrottleMin`
- `SndBufSize`
- `StackSize`
- `TerminateTimeout`
- `ThreadIncrement`
- `UseNativePoll` (Unix only)

For more information about these directives, see the *Application Server Developer's Guide to NSAPI*.

HTTP listener Acceptor Threads

You can specify how many threads you want in accept mode on a listen socket at any time. It is a good practice to set this to less than or equal to the number of CPUs in your system.

Tuning

You can set the number of HTTP listener acceptor threads by:

- Editing the `server.xml` file
- Selecting the `http-listener` node of the web-based Administration interface.

Maximum Simultaneous Requests

The `RqThrottle` parameter in the `init.conf` file specifies the maximum number of simultaneous transactions the web server can handle. The default value is 128. Changes to this value can be used to throttle the server, minimizing latencies for the transactions that are performed. The `RqThrottle` value acts across multiple virtual servers, but does not attempt to load-balance.

To compute the number of simultaneous requests, the server counts the number of active requests, adding one to the number when a new request arrives, subtracting one when it finishes the request. When a new request arrives, the server checks to see if it is already processing the maximum number of requests. If it has reached the limit, it defers processing new requests until the number of active requests drops below the maximum amount.

In theory, you could set the maximum simultaneous requests to 1 and still have a functional server. Setting this value to 1 would mean that the server could only handle one request at a time, but since HTTP requests for static files and generally have a very short duration (response time can be as low as 5 milliseconds), processing one request at a time would still allow you to process up to 200 requests per second.

However, in actuality, Internet clients frequently connect to the server and then do not complete their requests. In these cases, the server waits 30 seconds or more for the data before timing out. You can define this timeout period using the `AcceptTimeOut` directive in `init.conf`. The default value is 30 seconds. Also, some sites do heavyweight transactions that take minutes to complete. Both of these factors add to the maximum simultaneous requests that are required. If your site is processing many requests that take many seconds, you may need to increase the number of maximum simultaneous requests. For more information on `AcceptTimeOut`, see [AcceptTimeOut Information](#).

Suitable `RqThrottle` values range from 100-500, depending on the load.

`RqThrottleMin` is the minimum number of threads the server initiates upon start-up. The default value is 48. `RqThrottle` represents a hard limit for the maximum number of active threads that can run simultaneously, which can become a bottleneck for performance. The default value is 128.

NOTE If you are using older NSAPI plug-ins that are not re-entrant, they will not work with the multi-threading model described. To continue using them, revise them so they are re-entrant. If this is not possible, configure your server to work with them by setting `RqThrottle` to 1, and use a high value for `MaxProcs`, such as 48 or greater. This will adversely impact your server's performance.

Tuning

You can tune the number of simultaneous requests by:

- Editing `RqThrottleMin` and `RqThrottle` in the `init.conf` file.
- Entering the desired value in the Maximum Simultaneous Requests field from the Performance Tuning page under Preferences in the web-based Administration interface.

Improving Java Performance

There are a number of ways you can improve Java performance on the Application Server. These include:

- Using Alternate Thread Library
- Using Pre-compiled JSPs
- Configuring Class Reloading

Using an Alternate Thread Library on Solaris/Sparc

Using an alternate thread library such as `libthread` or `/usr/lib/lwp`, gives optimal performance. On Solaris 9, this `libthread` is enabled by default. On Solaris 8, you can enable it by following the instruction in [“Using Alternate Threads” on page 105](#).

Using Pre-compiled JSPs

Compiling JSPs is a resource intensive and time-consuming process. You will improve performance if you pre-compile your JSPs before installing them into your server.

Configuring Class Reloading

The configuration for flag for dynamic reloading should be disabled for better performance. This can be accomplished by editing the `server.xml` and setting `dynamic-reload-enabled="false"`

Miscellaneous `init.conf` Directives

The following sections discuss `init.conf` directives you can use to configure your server to function more effectively:

- [AcceptTimeout Information](#)
- [CGIStub Processes \(Unix\)](#)
- [Buffer Size](#)

AcceptTimeout Information

Use `AcceptTimeout` to specify the number of seconds the server waits between accepting a connection to a client and receiving information from it. The default setting is 30 seconds. Under most circumstances you should not have to change this setting. By setting it to less than the default 30 seconds, you can free up threads sooner. However, you may also disconnect users with slower connections.

Tuning

You can set the `AcceptTimeout` by:

- Editing the `AcceptTimeout` parameter in `init.conf`.
- Modifying the value in the performance sub-menu of the Advanced tab in the web-based Administration interface.

CGIStub Processes (Unix)

You can adjust the `CGIStub` parameters on Unix systems. In Application Server, the CGI engine creates `CGIStub` processes as needed. On systems that serve a large load and rely heavily on CGI-generated content, it is possible for the `CGIStub` processes to consume all system resources. If this is happening on your server, the `CGIStub` processes can be tuned to restrict how many new `CGIStub` processes can be spawned, their timeout value, and the minimum number of `CGIStub` processes that will be running at any given moment.

NOTE	If you have an <code>init-cgi</code> function in the <code>init.conf</code> file and you are running in multi-process mode, you must add <code>LateInit = yes</code> to the <code>init-cgi</code> line.
-------------	---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------

The four directives and their defaults that can be tuned to control CGI stubs are:

- `MinCGIStubs`
- `MaxCGIStubs`
- `CGIStubIdleTimeout`
- `CGIExpirationTimeout`

`MinCGIStubs` controls the number of processes that are started by default. The first `CGIStub` process is not started until a CGI program has been accessed. The default value is 2. If you have an `init-cgi` directive in the `init.conf` file, the minimum number of `CGIStub` processes are spawned at startup.

`MaxCGIStubs` controls the maximum number of `CGIStub` processes the server can spawn. This specifies the maximum concurrent `CGIStub` processes in execution, not the maximum number of pending requests. The default value shown should be adequate for most systems. Setting this too high may actually reduce throughput. The default value is 10.

`CGIStubIdleTimeout` causes the server to kill any `CGIStub` processes that have been idle for the number of seconds set by this directive. Once the number of processes is at `MinCGIStubs` it does not kill any more processes. The default is 45.

`CGIExpirationTimeout` limits the maximum time in seconds that CGI processes can run.

Tuning

You can configure all of the directives for CGI Stub processes by:

- Editing the `init.conf` file
- Modifying the value in the CGI sub-menu of the Advanced tab in the web-based Administration interface.

Buffer Size

You can specify the size of the send buffer (`SndBufSize`) and the receiving buffer (`RcvBufSize`) at the server's sockets. For more information regarding these buffers, see your Unix documentation.

Tuning

You can set the buffer size by:

- Editing the `SndBufSize` and `RcvBufSize` parameters in `init.conf`.
- Setting or changing the `SndBufSize` and `RcvBufSize` values in the Performance sub-menu of the Advanced tab in the web-based Administration interface.

Miscellaneous obj.conf Parameters

You can use some `<serverInstance>-obj.conf` function parameters to improve your server's performance. In addition to the ones listed below, see [“Using the nocache Parameter”](#) for additional information.

For more information on using `<serverInstance>-obj.conf`, see the *Application Server Developer's Guide to NSAPI*.

find-pathinfo-forward

The parameter `find-pathinfo-forward` for the `PathCheck` function `find-pathinfo` and the `NameTrans` functions `pfx2dir` and `assign-name` can help you improve your performance. This parameter instructs the server to search forward for `PATH_INFO` in the path after `ntrans-base`, instead of backward from the end of path in the server function `find-pathinfo`.

NOTE The server ignores the `find-pathinfo-forward` parameter if the `ntrans-base` parameter is not set in `rq->vars` when the server function `find-pathinfo` is called. By default, `ntrans-base` is set.

For example:

```
NameTrans fn="pfx2dir" find-pathinfo-forward="" from="/cgi-bin"
dir="/export/home/cgi-bin" name="cgi"

NameTrans fn="assign-name" from="/perf" find-pathinfo-forward=""
name="perf"
```

This feature can improve performance for certain URLs by doing fewer stats in the server function `find-pathinfo`.

nostat

You can specify the parameter `nostat` in the `NameTrans` function `assign-name` to prevent the server from doing a `stat` on a specified URL whenever possible. Use the following syntax:

```
nostat=virtual-path
```

For example:

```
<Object name=default>

NameTrans fn="assign-name" from="/nsfc" nostat="/nsfc" name="nsfc"

</Object>

<Object name=nsfc>

Service fn=service-nsfc-dump

</Object>
```

In the above example, the server does not `stat` for path `/ntrans-base/nsfc` and `/ntrans-base/nsfc/*` if `ntrans-base` is set. If `ntrans-base` is not set, the server does not `stat` for URLs `/nsfc` and `/nsfc/*`. By default `ntrans-base` is set. The example assumes the default `PathCheck` server functions are used.

When you use `nostat= virtual-path` in the `assign-nameNameTrans`, the server assumes that `stat` on the specified `virtual-path` will fail. Therefore, use `nostat` only when the path of the `virtual-path` does not exist on the system, for example, in NSAPI plug-in URLs. Using `nostat` on those URLs improves performance by avoiding unnecessary `stats` on those URLs.

Scaling Your Server

This section examines subsystems of your server and makes some recommendations for optimal performance:

- [Processors](#)
- [Memory](#)
- [Disk Space](#)
- [Networking](#)

Processors

On Solaris, the Application Server transparently takes advantage of multiple CPUs. In general, the effectiveness of multiple CPUs varies with the operating system and the workload. Dynamic content performance improves as more processors are added to the system. Because static content involves mostly IO, and more primary memory means more caching of the content (assuming the server is tuned to take advantage of the memory) more time is spent in IO rather than any busy CPU activity. Our study of dynamic content performance on a four-CPU machine indicate a 40-60% increase for NSAPI and about 50-80% increase for servlets., by doubling the number of CPUs.

Memory

The Application Server requires a minimum of 256 MB RAM on Solaris. These values apply to the application server running on a system that is not running Sun ONE Studio. Please refer to the *Application Server Installation Guide* on the Sun Microsystems documentation web site.

Disk Space

You need to have enough disk space for your OS, document tree, and log files. In most cases 2GB total is sufficient.

Put the OS, swap/paging file, Application Server logs, and document tree each on separate hard drives. Thus, if your log files fill up the log drive, your OS will not suffer. Also, you'll be able to tell whether, for example, the OS paging file is causing drive activity.

Your OS vendor may have specific recommendations for how much swap or paging space you should allocate. Based on our testing, Application Server performs best with swap space equal to RAM, plus enough to map the document tree.

Networking

For an Internet site, decide how many peak concurrent users you need the server to handle, and multiply that number of users by the average request size on your site. Your average request may include multiple documents. If you're not sure, try using your home page and all its associated sub-frames and graphics.

Next decide how long the average user will be willing to wait for a document, at peak utilization. Divide by that number of seconds. That's the WAN bandwidth your server needs.

For example, to support a peak of 50 users with an average document size of 24kB, and transferring each document in an average of 5 seconds, we need 240 KBs (1920 kbit/s). So our site needs two T1 lines (each 1544 kbit/s). This also allows some overhead for growth.

Your server's network interface card should support more than the WAN it's connected to. For example, if you have up to three T1 lines, you can get by with a 10BaseT interface. Up to a T3 line (45 Mbit/s), you can use 100BaseT. But if you have more than 50 Mbit/s of WAN bandwidth, consider configuring multiple 100BaseT interfaces, or look at Gigabit Ethernet technology.

For an intranet site, your network is unlikely to be a bottleneck. However, you can use the same calculations as above to decide.

Database drivers

It is always better to access database drivers using the classpath declared in `server.xml` classpath, rather than putting them in `<instance_dir>/lib`.

Classes in `<instance_dir>/lib` are loaded by the common class loader. But class accessed using the classpath in `server.xml` are loaded by the system-class loader. The system class loader is supposed to be the parent of common class loader. There are two reasons for preferring the system-class loader:

- A CMP beans application may get an exception, because the system class loader is used to access the driver.
- The server does additional performance optimizations for database access for classes which are loaded by the system-class loader.

Connection Pool Tuning

This section advises how users can tune their JDBC Connection Pools.

For database intensive applications, the JDBC Connection Pools managed by the Application Server can be tuned for optimum performance. These connection pools maintain numerous live physical database connections that can be reused in order reduce the overhead of opening and closing database connections.

JDBC Resources are defined as `<jdbc-resource>` elements in the Application Server configuration file `server.xml` and are configured to point to a `<jdbc-connection-pool>`. J2EE applications use JDBC Resources to obtain connections that are maintained by the JDBC Connection Pool. More than one JDBC Resource is allowed to point to the same JDBC Connection Pool. In such a case, the physical connection pool is shared by all the resources.

JDBC Connection Pools can be defined and configured by using the web-based Admin Console or by editing the `jdbcc-connection-pool` element in the `server.xml` file. Though each defined pool is instantiated at server start-up, the pool is only populated with physical connections when accessed for the first time.

The following are the attributes that can be specified for a JDBC connection pool:

Table 3-1 JDBC Connection Pool Attributes

Name	Description
name	Unique name of the pool definition.
datasource-classname	Name of the vendor supplied JDBC datasource resource manager. An XA or global transactions capable datasource class will implement <code>javax.sql.XADataSource</code> interface. Non XA or Local transactions only datasources will implement <code>javax.sql.DataSource</code> interface.
res-type	Datasource implementation class could implement one or both of <code>javax.sql.DataSource</code> , <code>javax.sql.XADataSource</code> interfaces. This optional attribute must be specified to disambiguate when a Datasource class implements both interfaces. An error is produced when this attribute has a legal value and the indicated interface is not implemented by the datasource class. This attribute has no default value.
steady-pool-size	Minimum and initial number of connections created.
max-pool-size	Maximum number of connections that can be created.
max-wait-time-in-millis	Amount of time the caller will wait before getting a connection timeout. The default is 60 seconds. A value of 0 will force caller to wait indefinitely.
pool-resize-quantity	Number of connections to be removed when <code>idle-timeout-in-seconds</code> timer expires. Connections that have idled for longer than the timeout are candidates for removal. When the pool size reaches <code>steady-pool-size</code> , the connection removal stops.
idle-timeout-in-seconds	Maximum time in seconds that a connection can remain idle in the pool. After this time, the pool implementation can close this connection. Note that this does not control connection timeouts enforced at the database server side. Administrators are advised to keep this timeout shorter than the database server side timeout (if such timeouts are configured on the specific vendor's database), to prevent accumulation of unusable connection in Application Server.
transaction-isolation-level	Specifies the Transaction Isolation Level on the pooled database connections. This setting is optional and has no default. If left unspecified the pool operates with default isolation level provided by the JDBC Driver. A desired isolation level can be set using one of the standard transaction isolation levels: <code>read-uncommitted</code> , <code>read-committed</code> , <code>repeatable-read</code> , <code>serializable</code> .

Table 3-1 JDBC Connection Pool Attributes

<code>is-isolation-level-guaranteed</code>	<p>Applicable only when a particular isolation level is specified for <code>transaction-isolation-level</code>. The default value is <code>true</code>.</p> <p>This assures that every time a connection is obtained from the pool, it is guaranteed to have the isolation set to the desired value.</p> <p>This setting can have some performance impact on some JDBC drivers. It can be set to <code>false</code> by that administrator when they are certain that the application does not change the isolation level before returning the connection.</p>
<code>is-connection-validation-required</code>	<p>If <code>true</code>, connections are validated (checked to find out if they are usable) before being given out to the application. Also, the <code>connection-validation-type</code> specifies the type of validation to be performed. The default is <code>false</code>. Types of validation supported:</p> <ol style="list-style-type: none"> 1) using <code>connection.setAutoCommit()</code>, 2) using <code>connection.getMetaData()</code> 3) performing a query on a user specified table (see <code>validation-table-name</code>). <p>The possible values are one of: <code>auto-commit</code>, or <code>meta-data</code>.</p> <p>The table <code>validation-table-name</code> attribute specifies the table name to be used to perform a query to validate a connection. This parameter is mandatory, if <code>connection-validation-type</code> is set to <code>table</code>. Verification by accessing a user specified table may become necessary for connection validation, particularly if database driver caches calls to <code>setAutoCommit()</code> and <code>getMetaData()</code>.</p>
<code>fail-all-connections:</code>	<p>Indicates if all connections in the pool must be closed should a single validation check fail. The default is <code>false</code>. One attempt will be made to re-establish failed connections.</p>

JDBC Connection Pool Tuning

The following performance tuning practices are recommended for JDBC Connection Pools:

- Use the default isolation level provided by the driver rather than calling the `setTransactionIsolationLevel()`; unless you are certain that your application behaves correctly and performs better at a different isolation level.
- Set the idle time out to 0 seconds. This directive ensures that the connections which are idle will not be removed at all. This ensures that there is normally no penalty in creating new connections and disables the idle monitor thread. However, there is a risk that the connection unused for too long is reset by the database server.
- When sizing connection pools, keep the following pros and cons in mind:

Table 3-2 Connection Pool Sizing Pros and Cons

Connection pool	Pros	Cons
Small Connection pool	<ul style="list-style-type: none">• faster access on the connection table.	<ul style="list-style-type: none">• not enough connections to satisfy requests.• most requests will spend more time in the queue.
Large Connection pool	<ul style="list-style-type: none">• more connections to fulfill requests.• less (or no) time in the queue	<ul style="list-style-type: none">• slower access on the connection table.

- Set the `max-wait-time-in-millis` to 0. This essentially blocks the client thread until a connection becomes available. Also, this allows the server to alleviate the task of tracking the elapsed wait time for each request and increases performance.
- As previously noted, avoid modifying the `transaction-isolation-level`. If that is not possible, consider setting the `is-isolation-level-guaranteed` flag to false and make sure applications do not programmatically alter the connections' isolation level.
- Setting the `is-connection-validation-required` parameter to true forces the server to apply the connection validation algorithm every time a connection is returned from the pool. This adds overhead to the latency of `getConnection()`. If the database connectivity is reliable, validation can be skipped.

Monitoring the JDBC Connection Pools

Statistics-gathering is enabled by default for JDBC Connection Pools. The following attributes are monitored:

- `numConnFailedValidation` (count)
The number of connections that failed validation.
- `numConnUsed` (range)
The number of connections that have been used.
- `numConnFree` (count)
The number of free connections in the pool.

- `numConnTimedOut` (bounded range)
The number of connections in the pool that have timed out.

To get the statistics, use these commands:

```
asadmin get --monitor=true
    serverInstance.resources.jdbc-connection-pool.*
asadmin get --monitor=true
    serverInstance.resources.jdbc-connection-pool. poolName.* *
```

JSP and Servlet Tuning

This section advises how users can tune JSP and Servlet applications by following several coding practices and checking several relevant Application Server configuration settings.

Suggested Coding Practices for JSP's and Servlets

The following coding practices are recommended for JSP and Servlet applications:

1. Do not store large objects as `HttpSession` variables.
2. Use `javax.servlet.http.HttpSession.invalidate()` to release HTTP Sessions when they are no longer needed.
3. Use the JSP directive `<%page session="false"%>` to prevent HTTP Sessions from being automatically created when they are not necessary.
4. Minimize Java synchronization in Servlets.
5. Don't use the single thread model for Servlets.
6. Use the servlet's `init()` method to perform expensive one time initialization.
7. Avoid the use of `System.out.println()` calls.

Configuration Settings that Affect JSP/Servlet Performance

The following configuration settings will improve performance. It is important to remember that they are intended for production environments as some of the settings make developing JSP and Servlets impractical.

1. In the server `CLASSPATH` setting, avoid excessive directories to improve class loading time. Package application related classes into JAR files.
2. HTTP settings - connections, keep-alive subsystem settings: the response times are dependent on how the keep-alive subsystem and the HTTP server is tuned in general. Please refer to the section on HTTP Server Tuning for more information.
3. Set the recompile reload interval to -1 to prevent JSP recompilation.
4. Use `mtmalloc` for SSL. Functions in this library provide a collection of `malloc` routines that provide concurrent access to heap space.

Obtain patch 111308-03 from <http://sunsolve.sun.com/> and install it. Edit the `startserv` script located in `bin/startserv` for your domain, and add the following code at the beginning of the file:

```
LD_PRELOAD=/usr/lib/libmtmalloc.so
export LD_PRELOAD
```

5. JSP Servlet caching configuration. For more information, see the *Application Server Developer's Guide to Web Applications* specifically, the chapter titled *Using Servlets* subsection on *Caching Features*.
6. Deploy applications that do not contain EJB's as a WAR file rather than an EAR file.
7. The security manager is expensive. This is because all the calls to the required resources will have to go through a `doPrivileged()` method call. This also involves checking the resource in question with the `server.policy` file. If there is an option where having `server.policy` doesn't make sense for the application, and under the assumption that no malicious code will be run on the server, then the user can disable the `server.policy` by commenting out the line in `server.xml`.

For example, you can comment out the `server.policy` as follows:

```
<!-- jvm-options>
-Djava.security.policy=/export/home/software/ias70_gold3/domains/domain
1/server1/config/server.policy
</jvm-options -->
```

Performance Tuning for EJBs

The Application Server's high performance EJB Container provides various tunables, with default values, that can be modified in the `server.xml` configuration file, and in each bean's descriptors. The values in `server.xml` apply to all EJBs unless they are also specified in the bean's deployment descriptors. Properties set in a bean's descriptors always override any settings in the `server.xml`. For a detailed description of `<ejb-container>` element in the `server.xml` file, see the *Application Server Configuration File Reference*.

Some properties that are in the EJB 2.0 deployment descriptor are also a good source of tuning. The default settings for the `<ejb-container>` element in the `server.xml` file are set for a single processor computer system. A user may want to change the default settings in order to derive the desired behavior from the container. The desired effects after tuning are for:

- **Speed:** Response times can be decreased effectively by caching as many beans in the EJB caches as possible. This saves several CPU-intensive operations (as explained below). However, one can have only finite memory as a resource, as the caches become large, house-keeping for these caches (including garbage collection) takes longer.
- **Memory consumption:** Beans in the pools or caches consume memory from the Java Virtual Machine heap. Exceedingly large pools and caches are detrimental to performance due to longer and frequent garbage collection cycles.
- **Functional properties:** (such as user timeout, commit options, security and transaction options, etc.) - Most of these properties are related to the functionality and configuration of the application in the J2EE server. It is not recommended to compromise functionality for performance, though some suggestions will be made to help make choices in case such a situation arises.

Monitoring EJBs and Containers

EJB statistics are disabled by default. To gather statistics, enable monitoring with these commands:

```
set serverInstance.application.application
    .ejb-module.module.monitoringEnabled=true
reconfig serverInstance
```

NOTE Monitoring can be enabled and disabled selectively at different levels in the monitoring hierarchy, shown below.

To gather method-invocation statistics for all methods in a bean, use this command:

```
asadmin get -m monitorableObject.*
```

where *monitorableObject* is a fully-qualified identifier from the hierarchy of monitorable objects, as shown below.

```
serverInstance.application.applicationName
```

- .ejb-module.moduleName (for x.jar, moduleName is x_jar)
 - .stateless-session-bean.beanName
 - .bean-pool
 - .bean-method.methodName
 - .stateful-session-bean.beanName
 - .bean-cache
 - .bean-method.methodName
 - .entity-bean.beanName
 - .bean-cache
 - .bean-pool
 - .bean-method.methodName
 - .message-driven-bean.beanName
 - .bean-pool
 - .bean-method.methodName (methodName = onMessage)

For standalone beans, use this pattern:

```
serverInstance.application.applicationName
```

- .standalone-ejb-module.moduleName
(same possibilities as for ejb-module, above)

For example, to get statistics for a method in an entity bean, use this:

```
asadmin get -m serverInstance.application.appName.ejb-module
    .moduleName.entity-bean.beanName.bean-method.methodName.*
```

To find the possible objects (applications, modules, beans, and methods) and object attributes that can be monitored, use the Admin Console or the `asadmin list` command, as described in the *Administration Guide* chapter on monitoring the Application Server.

For statistics on stateful session bean passivations, use this command:

```
asadmin get -m serverInstance.application.appName.ejb-module
               .moduleName.stateful-session-bean.beanName.bean-cache.*
```

From the attribute values that are returned, use the following:

```
num-passivations
num-passivation-errors
num-passivation-success
```

Tuning the EJB Container

The EJB specification formally defines the life cycle of various types of beans. This document assumes that you are familiar with bean lifecycle events. Active beans in the container process requests and are cached or pooled for better performance. Tuning the cache and pool properties is a significant part of tuning for performance.

Depending on the type of a bean, some of the suggested tuning tips may not apply to a particular container.

Guide to Using Tunables

The following table illustrates the cache and bean tunables for each type of EJB.

Table 3-3 Cache and Bean Tunables for EJBs

	Cache Tunables						Pool Tunables			
Type of Bean	cache-resize-quantity	max- cache-size	cache-idle-timeout-in-seconds	removal-timeout-in-seconds	victim-selection-policy	refresh-period-in-seconds	steady-pool-size	pool-resize-quantity	max-pool-size	pool-idle-timeout-in-seconds
Stateful Session	X	X	X	X	X					
Stateless Session							X	X	X	X
Entity (BMP/CMP)	X	X	X	X	X		X	X	X	X
Entity (BMP) ReadOnly	X	X	X	X	X	X	X	X	X	X
Message Driven Bean								X	X	X

EJB Descriptor Properties

The following properties are available to tune for each bean in EJB Container:

- `steady-pool-size`: `steady-pool-size` specifies the initial and minimum number of beans that must be maintained in the pool. Valid values are from 0 to `MAX_INTEGER`. This value is specified at the server instance level. The bean specific counterpart to this is also the same. Note that for all the variables below, if there is a setting specified at the bean level (in the `sun-ejb-jar.xml`), the values specified at the bean level are used.

- `pool-resize-quantity: pool-resize-quantity` specifies the number of beans to be created or deleted when the pool is being serviced by the server. Valid values are from 0 to `MAX_INTEGER` and subject to maximum size limit). Default is 16. The corresponding attribute in the `sun-ejb-jar.xml` is `resize-quantity`.
- `max-pool-size: max-pool-size` specifies the maximum pool size. Valid values are from 0 to `MAX_INTEGER`. Default is 64. A value of 0 means that the size of the pool is unbounded. The potential implication is that the JVM heap will be filled with objects in the pool. The corresponding attribute in the `sun-ejb-jar.xml` is `max-pool-size` in the `<bean-pool>` element.
- `max-wait-time-in-millis` (deprecated)
- `pool-idle-timeout-in-seconds: pool-idle-timeout-in-seconds` specifies the maximum time that a stateless session bean, entity bean, or message-driven bean is allowed to be idle in the pool. After this time, the bean is destroyed if the bean in case is a stateless session bean or a message driver bean. This is a hint to server. Default value for `pool-idle-timeout-in-seconds` is 600 seconds. The corresponding attribute in the `sun-ejb-jar.xml` is `pool-idle-timeout-in-seconds` in the `<bean-pool>` element.
- `cache-resize-quantity: cache-resize-quantity` specifies the number of beans to be created or deleted when the cache is being serviced by the server. Valid values are from 0 to `MAX_INTEGER` and subject to maximum size limit. Default is 16. The corresponding attribute in the `sun-ejb-jar.xml` is `resize-quantity` in the `<bean-cache>` element.
- `max-cache-size: max-cache-size` defines the maximum number of beans in the cache. Should be greater than 1. Default is 512. A value of 0 indicates the cache is unbounded. The size of the cache, in this case, is governed by `cache-idle-timeout-in-seconds` and `cache-resize-quantity`. The corresponding attribute in the `sun-ejb-jar.xml` is `max-cache-size` in the `<bean-cache>` element.
- `cache-idle-timeout-in-seconds: cache-idle-timeout-in-seconds` specifies the maximum time that a stateful session bean or entity bean is allowed to be idle in the cache. After this time, the bean is passivated to backup store. Default value for `cache-idle-timeout-in-seconds` is 600 seconds. The corresponding attribute in the `sun-ejb-jar.xml` is `cache-idle-timeout-in-seconds` in the `<bean-cache>` element.
- `is-cache-overflow-allowed` (deprecated)

- `removal-timeout-in-seconds`: The amount of time that a stateful session bean remains passivated (i.e. idle in the backup store) is controlled by `removal-timeout-in-seconds` parameter. Note that if a bean was not accessed beyond `removal-timeout-in-seconds`, then it will be removed from the backup store and hence will not be accessible to the client. The Default value for `removal-timeout-in-seconds` is 60min. The corresponding attribute in the `sun-ejb-jar.xml` is `removal-timeout-in-seconds` in the `<bean-cache>` element.
- `victim-selection-policy`: `victim-selection-policy` specifies the algorithm to use to pick victims to be removed from the stateful session bean cache. Possible values are `FIFO` | `LRU` | `NRU`. Default is `NRU`, which is actually pseudo-random selection policy. The corresponding attribute in the `sun-ejb-jar.xml` is `victim-selection-policy` in the `<bean-cache>` element.
- `commit-option`: values are either "B", or "C". Default is "B". These reflect `commit-options` for transactions as per the EJB specification.
- `refresh-period-in-seconds`: (for BMP/ Read Only Beans only) `refresh-period-in-seconds` specifies the rate at which the read-only-bean must be refreshed from the data source. 0 (never refreshed) and positive (refreshed at specified intervals). Default is 600 seconds.

Tuning the EJB Pool

A bean in the pool represents the pooled state in the EJB lifecycle. This means that the bean does not have an identity. The advantage of having beans in the pool is that the time to create a bean may be saved for a request. The container has mechanisms that create pool objects in the background, to save the time of bean creation on the request path.

Set the `steady-pool-size` to a number that is indicative of a moderately loaded system. It is recommended that `steady-pool-size` be greater than 0, as it ensures that there is always a pooled instance to process an incoming request.

Set the `max-pool-size` to be representative of the anticipated high load of the system. An excessively large pool wastes memory and can slow down the system. A very small pool is also inefficient due to contention.

A good rule to remember when changing the `<max-pool-size>` is to also re-calibrate the `<pool-resize-quantity>`. This quantity is the number of beans that will be reclaimed by the periodic cleaner. An increase in the max size should mean an appropriate increase in the resize quantity to maintain a good equilibrium.

Another important tunable is the `<pool-idle-timeout-in-seconds>` value. In case there are more beans in the pool than the `<steady-pool-size>`, the pool will be drained back to `<steady-pool-size>` in steps of `<pool-resize-quantity>`, every `<pool-idle-timeout-in-seconds>` seconds. If the resize quantity is too small and the idle timeout large, the user would not see the pool draining back to steady size quick enough; this should be expected or corrected.

Tuning the EJB Cache

A bean in the cache represents the ready state in the EJB lifecycle. This means that the bean has an identity (e.g. primary key or session ID) associated with it. Beans moving out of the cache have to be passivated or destroyed according to the EJB lifecycle. Once passivated, a bean has to be activated to come back into the cache. Entity beans are generally stored in databases and use some form of query language semantics to load and store data. Session beans have to be serialized when storing them upon passivation onto the disk or a database; and similarly have to be deserialized upon activation.

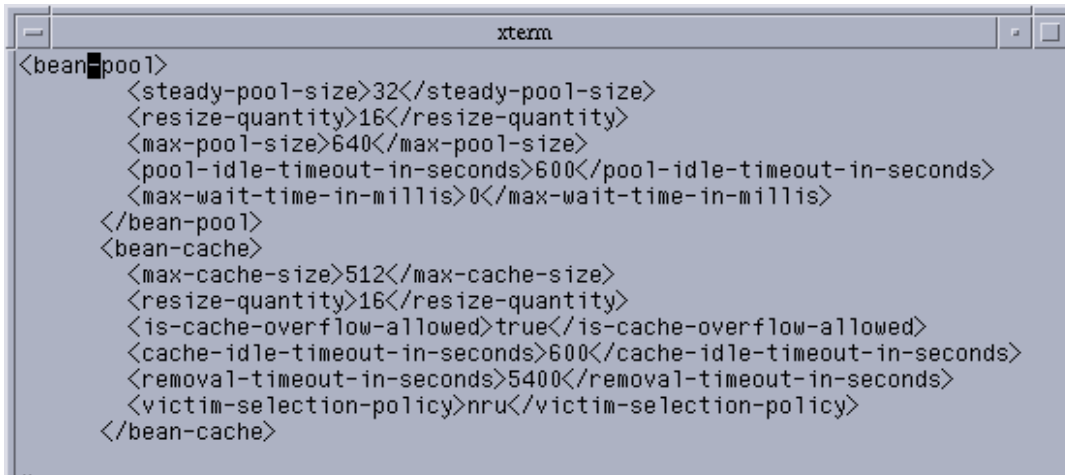
Any incoming request using these 'ready' beans from the cache avoids the overheads of creation, setting identity, and potentially, activation. So, theoretically, it is good to cache as many beans as possible. However, there are downsides to caching extensively:

- memory consumed by all the beans affect the heap available in the Virtual Machine,
- increasing objects and memory taken by cache means longer, and perhaps more frequent, full Garbage Collection,
- application server might run out of memory (unless the heap is carefully tuned for peak loads).

The periodic cleaner will remove all beans in the cache that have reached the `<cache-idle-timeout-in-seconds>`.

Performance Considerations for Various Types of EJBs

The following figure shows an example of possible bean descriptors for an individual bean pool and cache.



```

xterm
<bean-pool>
  <steady-pool-size>32</steady-pool-size>
  <resize-quantity>16</resize-quantity>
  <max-pool-size>640</max-pool-size>
  <pool-idle-timeout-in-seconds>600</pool-idle-timeout-in-seconds>
  <max-wait-time-in-millis>0</max-wait-time-in-millis>
</bean-pool>
<bean-cache>
  <max-cache-size>512</max-cache-size>
  <resize-quantity>16</resize-quantity>
  <is-cache-overflow-allowed>true</is-cache-overflow-allowed>
  <cache-idle-timeout-in-seconds>600</cache-idle-timeout-in-seconds>
  <removal-timeout-in-seconds>5400</removal-timeout-in-seconds>
  <victim-selection-policy>nru</victim-selection-policy>
</bean-cache>

```

Figure 3-5 Bean Descriptors for Individual

The following are performance related discussions of various bean types:

- Entity beans:** Depending on the usage of a particular entity bean, one should tune max-cache-size so that beans that are used less (e.g. an order that is created and never used after the transaction is over) are cached less, and beans that are used frequently (e.g. an item in the inventory that gets referenced very often), are cached more in numbers. Please see the section titled [Commit Options](#) for other ways of tuning the entity container.
- Stateful session beans:** In case a stateful bean represents a user, a healthy max-cache-size of beans could be the expected number of concurrent users on the application server process. If this value is too low (in relation to the steady load of users), beans would be frequently passivated and activated, causing a negative impact on the response times, due to CPU intensive serialization and deserialization as well as disk I/O. Another important variable for tuning is cache-idle-timeout-in-seconds where at periodic intervals of cache-idle-timeout-in-seconds, all the beans in the cache that have not been accessed for more than cache-idle-timeout-in-seconds time, are passivated. Similar to an HttpSession inactivity/ idle timeout, the bean is removed after it has not been accessed for removal-timeout-in-seconds. Passivated beans are stored on disk in serialized form. A large number of passivated beans could not only mean many files on the disk system, but also slower response time as the session state has to be deserialized before the invocation.

- **Stateless session beans:** There is no state associated with stateless session beans and they are more readily pooled than entity or the stateful session beans. Appropriate values for `steady-pool-size`, `pool-resize-quantity` and `max-pool-size` are the best tunables for these type of beans. Set the `steady-pool-size` to greater than zero if you want to pre-populate the pool. This way, when the container comes up, it creates a pool with `steady-pool-size` number of beans. By pre-populating the pool you can avoid the object creation time during method invocations. Setting the `steady-pool-size` to a very large value may cause unwanted memory growth and may result in large GC times. `pool-resize-quantity` determines the rate of growth as well as the rate of decay of the pool. Setting it to a small value will be better as the decay will behave like an exponential decay. Setting a small `max-pool-size` may cause excessive object destruction (and as a result excessive object creation) as instances will be destroyed from the pool if the current pool size exceeds `max-pool-size`.
- **Read only entity beans:** If you have a regular BMP bean that never updates the database, try using a read-only bean in its place. By default, a read-only bean will work correctly only if the database row(s) represented by this bean are not being changed in the background. In the Application Server, read-only beans are supported only for bean managed persistence (BMP). A read-only bean never updates the database (i.e., `ejbStore` is never called). Consequently, there is a significant performance gain between a BMP bean and a read-only bean. If a read-only bean's method is accessed with a transaction (because the method descriptor in `ejb-jar.xml` is `TX_REQUIRED` or `TX_REQUIRES_NEW`), the `ejbLoad()` is always called. You may want to access a read-only bean method with a transaction if you want the cached data to always be in sync with the database. Its the non-transactional accesses where read-only beans perform at their best by avoiding the overhead of `ejbLoad()`. An important tunable to consider in this case is `refresh-period-in-seconds`. The current implementation of the container for these beans is a pull-cache, i.e. if for any beans instance, more than `refresh-period-in-seconds` time has passed, a user access to this bean instance will first invoke a `ejbLoad()` on the bean before executing any business methods. Tune this time to an appropriate value, typically in minutes, where you anticipate the data to change. A good example is when hosting a stock quote with 5 minute periods, i.e. 300 seconds set as the `refresh-period-in-seconds`.

NOTE If you have a regular BMP bean that updates the data cached by a read-only bean, then consider using the programatic refresh feature described in the *Application Server Developer's Guide to EJB Technology*.

NOTE	If the beans are developed and deployed onto the application server using Sun ONE Studio, then the user needs to edit the individual bean descriptor settings for bean pool and bean cache. These settings may not be suitable for production level deployment.
-------------	-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------

- **Message driven beans:** The container of these beans is a bit different than the containers for entity and session beans. In the MDB container, session and threads are attached to the beans in the MDB pool. That design makes it possible to pool the threads for executing message driven requests in the container. Thus the bean pool should be given an appropriate value, considering all the parameters of the server (taking other applications into perspective). e.g. values greater than 500 will be inappropriate.

Related EJB Considerations

Here are related considerations to keep in mind when using EJB's.

- Remote vs Local interfaces

An EJB can have remote and local interfaces. Remote clients which are not co-located in the same application server instance use the remote interface to access the bean. Calls to the remote interface are more expensive as it involves argument marshalling, transportation of the marshalled data over the network and unmarshalling and dispatch at the receiving end. Local clients which are co-located in the same application server instance can use the local interface of a bean if provided. Using the local interface is more efficient as it does not involve any argument marshalling, transportation and unmarshalling. The arguments are passed by reference and this means that the passed objects are shareable by the caller and callee (hence they need to be implemented appropriately so that they are shareable). If a bean is written to be used only by co-located clients then it makes sense to just provide the local interface for the bean and have the clients use the local interface. If on the other hand the bean is to be written in a location independent manner then both the remote and local interface can be provided so that remote clients use the remote interface and local clients can use the local interface for efficiency.

- Pass-by-value vs Pass-by-reference semantics

Using the remote and local interface appropriately it should be possible to write beans that can be accessed by clients efficiently. But there may be some instances where it may not be possible to use the local interface:

1. The application predates EJB 2.0 specification and was written without any local interfaces.
2. There are bean to bean calls and the client beans are written without making any co-location assumptions about the called beans.

For these cases the Application Server provides a pass-by-reference option which can be used to indicate that when making calls to the remote interface of the bean a co-located client can pass arguments by reference. By default the Application Server uses pass-by-value semantics for calling the remote interface of a bean even if it is co-located. This can be quite expensive as pass-by-value semantics involves making copies of the argument before passing them. The pass-by-reference option can be specified at the entire application level or on a per bean basis. If it is specified at the application level then pass-by-reference semantics is used when passing arguments to the remote interface of all beans in the application. If this option is specified at a bean level then calls to the remote interface of the specified bean use the pass-by-reference semantics. Please refer to the *EJB Developers Guide* and *Application Server Developers Guide* for more details about the pass-by-reference flag.

- Transaction isolation levels:

With a clear idea on the application semantics with respect to transactions, one can choose the correct isolation level for optimum performance. The following are the transaction isolation levels listed in the order of performance from best for performance to worst:

- a. `READ_UNCOMMITTED`
- b. `READ_COMMITTED`
- c. `REPEATABLE_READ`
- d. `SERIALIZABLE`

These values can be specified as a attribute of the database connection pool (`jdbc-connection-pool`).

3. From the EJB container point of view, the following transaction attributes can be specified again from the following list. Options are listed in the order that is best for performance to worst.
 - a. `NEVER`
 - b. `TX_NOTSUPPORTED`
 - c. `TX_MANDATORY`

- d. TX_SUPPORTS
- e. TX_REQUIRED
- f. TX_REQUIRESNEW

Commit Options

Commit option controls the action taken by the container on a bean when the transaction that the bean participated completes. Commit option has no effect on the bean code (the bean developer need not worry about the commit options). Commit option, however, has a significant impact on performance.

The Application Server supports commit option B and commit option C.

Before we explain when to use the various commit options, let us describe what the container does when these commit options are used.

In Commit option B, when a transaction completes, the bean is kept in the cache and retains its identity. This means that the next invocation for the same primary key can use the instance that is in the cache. Of course, the bean's `ejbLoad` will be called before the method invocation to sync up with the database.

In case of Commit option C, when a transaction completes, the bean's `ejbPassivate()` method is called, then the bean is disassociated from its primary key and then it is returned to the free pool. This means that the next invocation for the same primary key will have to grab a free bean from the pool, set the `PrimaryKey` on this instance, and then call `ejbActivate` on the instance. Again, the bean's `ejbLoad` will be called before the method invocation to sync up with the database.

It is clear that Commit Option B avoids `ejbActivate` and `ejbPassivate` calls. So, in most cases commit option-B should perform better than commit option-C since it avoids `ejbActivate`, `ejbPassivate` and some overhead in acquiring and releasing objects back to pool. However, there are some cases where commit option-C can do better. If the beans in the cache are very rarely reused and if beans are constantly added to the cache, then it makes no sense to cache beans.

Commit option C does exactly that. When commit option C is used, the container puts beans back into the pool (instead of caching the instance) after method invocation or on transaction completion. This way instances are reused better and the number of live objects in the VM is reduced resulting in smaller GC cycle.

How would you decide whether to use Commit option B or commit option C? First take a look at the cache-hits value using the monitoring command for the bean. If the cache-hits are very high compared to cache-misses, then commit-B option is an appropriate choice. You may still have to change the `max-cache-size` and `cache-resize-quantity` to get the best result. If the cache hits are too low and cache

misses are very high, then the application is not reusing the bean instances and hence increasing the cache size (using `max-cache-size`) will not help (assuming that the access pattern remains the same). In this case you may want to use commit option-C. If there is no great difference between cache-hits and cache-misses then you may have to tune `max-cache-size`, and probably `cache-idle-timeout-in-seconds`.

The following figure shows the commit option settings.

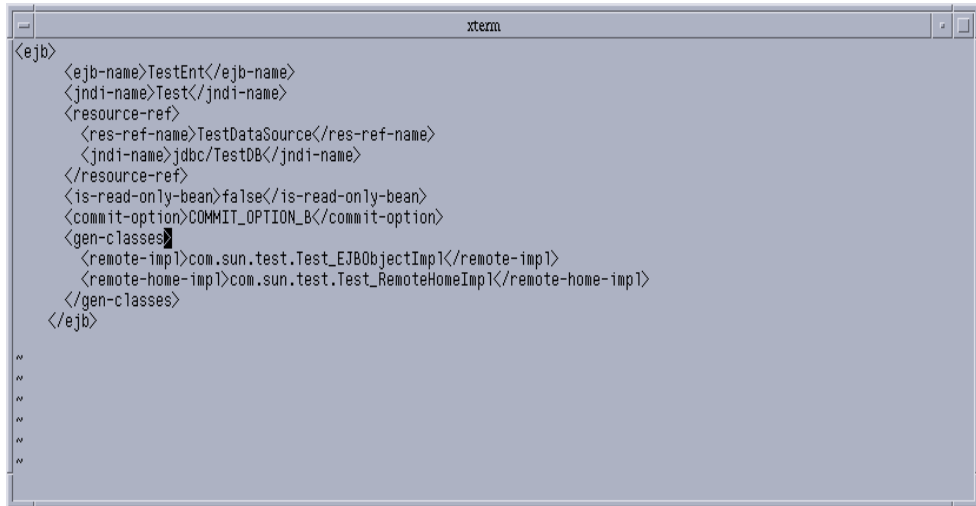


Figure 3-6 Commit Option Settings

At any given instance of time, if monitoring is enabled for `ejb-container`, the statistics for the individual beans can be examined and analyzed based on the bean pool and cache settings can be tuned. The pool settings are valid for stateless session and entity beans while the cache settings are valid for stateful session and entity beans. The configuration for the container can be done at the server instance level, via setting the properties in the `server.xml` file. These values can be overwritten by specifying the values for the individual beans in the `sun-ejb-jar.xml`. For a description of the tunables below, please refer to the EJB Descriptor Properties section.

The settings that can be specified at the server instance level are:

- `steady-pool-size`
- `pool-resize-quantity`
- `max-pool-size`

- cache-resize-quantity
- max-cache-size
- pool-idle-timeout-in-seconds
- cache-idle-timeout-in-seconds
- removal-timeout-in-seconds
- victim-selection-policy
- commit-option
- log-level
- monitoring-enabled

The settings for the pool that can be specified at the bean level are:

- steady-pool-size
- resize-quantity
- max-pool-size
- pool-idle-timeout-in-seconds

The settings for cache that can be specified at the bean level are:

- max-cache-size
- resize-quantity
- is-cache-overflow-allowed
- cache-idle-timeout-in-seconds
- removal-timeout-in-seconds
- victim-selection-policy

The monitoring command below gives the Bean Cache statistics for a stateful session bean. The following is a sample of the monitoring output:

```
$. /asadmin get --user admin --password netscape --host e4800-241-a --port 4848 -m specjcmp.application.SPECjAppServer.ejb-module.supplier_jar.stateful-session-bean.BuyerSes.bean-cache.*  
  
resize-quantity = -1  
  
cache-misses = 0
```

```

idle-timeout-in-seconds = 0
num-passivations = 0
cache-hits = 59
num-passivation-errors = 0
total-beans-in-cache = 59
num-expired-sessions-removed = 0
max-beans-in-cache = 4096
num-passivation-success = 0

```

The monitoring command below gives the bean pool statistics for a Entity bean:

```

$./asadmin get --user admin --password netscape --host e4800-241-a --port
4848 -m
specjcmp.application.SPECjAppServer.ejb-module.supplier_jar.stateful-entit
y-bean.ItemEnt.bean-pool.*
idle-timeout-in-seconds = 0
steady-pool-size = 0
total-beans-destroyed = 0
num-threads-waiting = 0
num-beans-in-pool = 54
max-pool-size = 2147483647
pool-resize-quantity = 0
total-beans-created = 255

```

The monitoring command below gives the bean pool statistics for a stateless bean.

```

$./asadmin get --user admin --password netscape --host e4800-241-a --port
4848 -m
test.application.testEjbMon.ejb-module.slsb.stateless-session-bean.slsb.be
an-pool.*
idle-timeout-in-seconds = 200
steady-pool-size = 32
total-beans-destroyed = 12
num-threads-waiting = 0
num-beans-in-pool = 4
max-pool-size = 1024

```

```
pool-resize-quantity = 12
```

```
total-beans-created = 42
```

Tuning the bean involves charting out the behavior of the cache and pool for the bean in question over a period of time. Some of the observations that can be made are:

- If too many passivations are happening and the VM heap remains fairly small, then the `max-cache-size` can be increased or the `cache-idle-timeout-in-seconds` can be increased.

If too many GCs are happening and the pool size is growing, but the cache hit rate is small, then the `pool-idle-timeout-in-seconds` can be reduced to destroy the instances.

NOTE Specifying the `max-pool-size` as 0 means that the pool is unbounded. The pooled beans remain in memory unless they are removed by specifying a small interval for the `pool-idle-timeout-in-seconds`. For production systems, specifying the pool as unbounded is NOT recommended.

ORB Tuning

The Application Server includes a high performance and scalable CORBA ORB (Object Request Broker). The ORB is the foundation of the EJB Container on the server. Most of the functionality of the ORB is utilized when exercising Enterprise Java Beans via:

1. RMI/ IIOP path from an application client (or rich client) using Application client container.
2. RMI/ IIOP path from another Application Server instance ORB.
3. RMI/ IIOP path from another vendor's ORB.
4. In-process path from the web/ MDB (message driven beans) container.

When a connection is made from a server instance to another server instance ORB, the first instance starts and acts as a client-side ORB. SSL over IIOP uses an optimized transport that is one of the fastest, and utilizes native implementations of cryptography algorithms to deliver high performance.

How a Client Connects to the ORB

A rich client Java program performs a new `initialContext()` call which creates a client side ORB instance. This in turn creates a socket connection to the Application Server IIOP port. The reader thread is started on the server ORB to service IIOP requests from this client. Using the `initialContext`, the client code does a lookup of an EJB deployed on the server. An IOR which is a remote reference to the deployed EJB on the server is returned to the client. Using this object reference, the client code invokes remote methods on the EJB.

`InitialContext` lookup for the bean and the method invocations translate the marshalling application request data in Java into IIOP message(s) that are sent on the socket connection that was created earlier on to the server ORB. The server then creates a response and sends it back on the same connection. This data in the response is then unmarshalled by the client ORB and given back to the client code for processing. The Client ORB shuts down and closes the connection when the rich client application exits.

Enabling the High performance CORBA Util Delegate

When using the JDK-bundled ORB or the Application Server ORB, you can get a significant increase in performance by enabling the high performance CORBA Util Delegate. To do so, add the following to the `server.xml` file:

```
<jvm-options>
  -Djavax.rmi.CORBA.UtilClass=com.ipplanet.ias.util.orbutil.IasUtilDeleg
  ate
</jvm-options>
```

Monitoring the ORB

ORB statistics are disabled by default. To gather ORB statistics, enable monitoring with these commands:

```
set serverInstance.iiop-service.orb.system.monitoringEnabled=true
reconfig serverInstance
```

The following statistics are gathered on ORB connections:

- `total-inbound-connections`
Total inbound connections to ORB.

- `total-outbound-connections`
Total outbound connections from ORB.

Use this command to get ORB connection statistics:

```
asadmin get --monitor
serverInstance.iiop-service.orb.system.orb-connection.*
```

The following statistics are gathered on ORB thread pools:

- `thread-pool-size`
Number of threads in ORB thread pool.
- `waiting-thread-count`
Number of thread pool threads waiting for work to arrive.

Use this command to get ORB thread pool statistics:

```
asadmin get --monitor
serverInstance.iiop-service.orb.system.orb-thread-pool.*
```

Tuning the ORB

You may want to change the default setting, as well as add some non-standard options to maximize performance and scalability. The main components of the ORB that can be tuned are:

- Inter-ORB Communication Infrastructure
- Server ORB thread pool

Response time can be decreased by leveraging load-balancing, multiple shared connections, finely tuned server thread pool and message fragment size. Scalability can be achieved by using multiple ORB servers and load balancing between them from the client, and tuning the number of connection between the client and the server(s).

ORB Tunables

The following sets of tunables are available on the ORB:

1. **Inter-ORB Communication Infrastructure:** The infrastructure allows for tuning the message size, load balancing (in cases of heavy load), better throughput, and high performance.

2. **Server ORB Thread Pool:** The ORB thread pool facilitates quick and simultaneous job execution through configuration-controlled multi-threading. Pooling threads mean that one can avoid overheads such as thread creation, thread stack allocation, associated GC, etc. In some cases, excessive thread creation and removal can lead to `OutOfMemoryError`, which the thread pool prevents, by providing thresholds.

The ORB thread pool contains a task queue and a pool of threads. Tasks or jobs are inserted into the task queue and free threads pick tasks from this queue for execution. It is not advisable to always size a thread pool size such that the task queue is always empty. It is normal for an intense application to have 1:10 ratio of 'current task queue size': `max-thread-pool-size` at any time. The thread pool has capability to shrink to the steady size if current size is larger and when `max-thread-pool-size > steady-thread-pool-size` is the configured setting. The `steady-thread-pool-size` should be set to the average number of threads needed at a steady (RMI/ IIOP) load.

In the current Application Server version, the ORB thread pool is used in two main activities:

1. execution of every ORB request, and
2. trimming of EJB pools and caches.

Thus even when one is not using ORB for remote-calls (i.e. via RMI/ IIOP), the thread pool should be sized so that cleaning-up activity of the EJB pools and caches can be facilitated.

ORB Properties

The properties for tuning the ORB can be managed using the Administration interface.

The following standard properties are available to tune on the ORB:

- `message-fragment-size`: CORBA GIOPv1.2 messages (15.4.9) larger than this size in bytes will be fragmented. All sizes should be multiples of 8. In GIOP v1.2, a Request, Reply, LocateRequest and LocateReply message can be broken into multiple fragments. The first message is a regular Request or Reply message with more fragments bit in the flags field set to true. When inter-ORB messages are mostly larger than the default size (1024 bytes), one can increase this size to decrease latencies on the network due to fragmentation.
- `steady-thread-pool-size`: The minimum number of threads in the ORB thread pool, should be close to the active number of thread needed to work on requests (and EJB cleanup) at server's steady state (i.e. in steady load conditions).

- `max-thread-pool-size`: The maximum number of threads in the ORB thread pool.
- `idle-thread-timeout-in-seconds`: Timeout when an idle thread is removed from pool. Allows shrinking of the thread pool.
- `max-connections`: Maximum number of incoming connections at any time, on all listeners. Protects the server state by allowing finite number of connections. This value equals also the the maximum number of threads that will be actively reading from the connection.
- `iiop-listener`: Property to add a listener (SSL/ HTTP) to the ORB, specifying the listener host and port. An enabled ORB listener translates to a thread actively listening on the server socket for incoming connection connect requests.

Non-standard ORB Properties and Functionality

The following values are specified as `-D` arguments when launching the client program:

Controlling connections between client and server ORB

When using the default JDK ORB on the client, a connection is established from the client ORB to the application server ORB every time an initial context is created. One may wish to pool or share these connections when they are opened from the same process by adding to the configuration on the client ORB -

```
-Djava.naming.factory.initial=com.sun.appserv.naming.SLASCtxFactory
```

Using multiple connections for better throughput

When using the Sun One context factory, (`com.sun.appserv.naming.SLASCtxFactory`) an important tunable is to specify the number of connections to open to the server from the client ORB (default is 1). This feature is seen to produce better throughput to and from the server for network intense application traffic. The configuration changes are specified on the client ORB(s) by adding the following jvm-options:

```
-Djava.naming.factory.initial=com.sun.appserv.naming.SLASCtxFactory
```

```
-Dcom.sun.appserv.iiop.orbconnections=[number]
```

Load Balancing

To configure RMI/IIOP for multiple application server instances in a cluster, refer to the *Application Server Administration Guide* chapter on RMI/IIOP Load Balancing.

High performance CORBA Util Delegate class

When using JDK-bundled ORB or Application Server ORB, users can benefit from a high performance CORBA Util Delegate implementation, that can be used by adding to the configuration (`server.xml`)

```
<jvm-options>-Djavax.rmi.CORBA.UtilClass=com.ipplanet.ias.util.orbutil.IasUtilDelegate</jvm-options>
```

An important consideration when tuning the client ORB for load-balancing and connections, is to consider the number of connections opened on the server ORB. It is always advisable to start from a lower number of connections and grow upwards to observe any performance benefits. A connection to the server translated to an ORB thread reading actively from the connection (these threads are not pooled, but exist currently for the lifetime of the connection).

Guide to using Tunables

The following table helps you identify the ORB modules and Server tunables involved in tuning your application.

Table 3-4 Using Tunables

Path	ORB modules involved	Tunables involved on server
RMI/ IIOP from application client to application server	communication infrastructure, thread pool	steady-thread-pool-size, max-thread-pool-size, idle-thread-timeout-in-seconds
RMI/ IIOP from Sun ONE (server) ORB to Application Server	communication infrastructure, thread pool	steady-thread-pool-size, max-thread-pool-size, idle-thread-timeout-in-seconds
RMI/ IIOP from a vendor ORB	parts of communication infrastructure, thread pool	steady-thread-pool-size, max-thread-pool-size, idle-thread-timeout-in-seconds
In-process	thread pool	steady-thread-pool-size, max-thread-pool-size, idle-thread-timeout-in-seconds

Thread Pool Sizing

After examining the number of inbound and outbound connections as explained above, users can tune the size of the thread pool appropriately. This can affect performance and response times to a large degree.

The size computation should take into account the number of client requests to be processed concurrently, the resource (number of cpus/memory) available on the machine and the response times required for processing the client requests. Setting the size to a very small value can affect the ability of the server to process requests concurrently thus affecting the response times of requests as they will be sitting longer in the task queue waiting for a worker thread to process it. On the other hand having a large number of worker threads to service requests can also be detrimental because more system resources are used up because of the large number of threads, which- increases concurrency. This can mean that threads take longer to acquire shared structures in the EJB container, thus affecting response times. The Worker thread pool is also used for the EJB containers house keeping activity such as trimming the pools and caches. This activity needs to be accounted for also when determining the size.

Having too many ORB worker threads is detrimental for performance since the server has to maintain all these threads. The idle threads are destroyed after the `idle-thread-time-out-in-seconds`. Below is a snippet from the `server.xml`. This includes the section for `iiop-service`.

```
<iiop-service>
  <orb message-fragment-size=1024
    steady-thread-pool-size=10
    max-thread-pool-size=200
    idle-thread-timeout-in-seconds=300
    max-connections=1024
    monitoring-enabled=false />
  <iiop-listener id=orb-listener-1 address=0.0.0.0 port=3700
    enabled=true>
</iiop-listener>
```

Related Considerations

Please refer to the [Tuning the EJB Container](#) section for Pass-by-value vs Pass-by-reference semantics considerations.

Examining IIOP Messages

It is sometimes useful to examine the contents of the IIOP messages being passed by the Application Server. The option `-Dcom.sun.CORBA.ORBDebug=giop` can be passed as jvm-options in `server.xml` to get dumps of the messages. The dumps are produced in `server.log`. The same option can also be used on the client ORB.

A sample output is as follows:

```
[29/Aug/2002:22:41:43] INFO (27179): CORE3282: stdout:
+++++

[29/Aug/2002:22:41:43] INFO (27179): CORE3282: stdout: Message(Thread[ORB
Client-side Reader, conn to 192.18.80.118:1050,5,main]):

createFromStream: type is 4 <

[29/Aug/2002:22:41:43] INFO (27179): CORE3282: stdout:
MessageBase(Thread[ORB Client-side Reader, conn to
192.18.80.118:1050,5,main]): Message GIOP version: 1.2

[29/Aug/2002:22:41:43] INFO (27179): CORE3282: stdout:
MessageBase(Thread[ORB Client-side Reader, conn to
192.18.80.118:1050,5,main]): ORB Max GIOP Version: 1.2

[29/Aug/2002:22:41:43] INFO (27179): CORE3282: stdout: Message(Thread[ORB
Client-side Reader, conn to 192.18.80.118:1050,5,main]): createFromStream:
message construction complete.

[29/Aug/2002:22:41:43] INFO (27179): CORE3282: stdout:
com.sun.corba.ee.internal.iiop.MessageMediator(Thread[ORB Client-side
Reader, conn to 192.18.80.118:1050,5,main]): Received message:

[29/Aug/2002:22:41:43] INFO (27179): CORE3282: stdout: ----- Input Buffer
-----

[29/Aug/2002:22:41:43] INFO (27179): CORE3282: stdout: Current index: 0

[29/Aug/2002:22:41:43] INFO (27179): CORE3282: stdout: Total length : 340

[29/Aug/2002:22:41:43] INFO (27179): CORE3282: stdout: 47 49 4f 50 01 02 00
04 0 0 00 01 48 00 00 00 05 GIOP.....H....
```

NOTE The flag `-Dcom.sun.CORBA.ORBdebug=giop` generates many debug messages in the logs. This should be used only when the user suspects message fragmentation.

Fragmented Messages

In this sample output above, the "createFromStream" type is shown as 4. This implies that the message is a fragment of a bigger message. One could change the fragment size to avoid fragmented messages. This would mean that messages would be sent as one unit and not as fragments thus saving the overhead of sending multiple messages and corresponding processing at the receiving end to piece the messages together. It might be more efficient to increase the fragment size if most messages being sent in the application turn out to be fragmented because of a low fragment size specification. On the other hand if only a few messages are fragmented, it might be more efficient to have a lower fragment size as this would mean smaller buffers would be allocated for writing messages and only the occasional message would end up getting fragmented.

Local Interfaces for EJB's

It is important to remember that the ORB is not used when using Local Interfaces for EJB's. In this situation, all arguments are passed by reference and no object copying is involved.

Transaction Manager Tuning

The transaction manager makes it possible to commit and roll back distributed transactions.

Monitoring the Transaction Manager

Transaction Manager are disabled by default. To gather statistics, enable monitoring with these commands:

```
set serverInstance.transaction-service.monitoringEnabled=true  
reconfig serverInstance
```

The following statistics are gathered on the transaction manager:

- `total-tx-completed`
Completed transactions.
- `total-tx-rolled-back`
Total rolled back transactions.
- `total-tx-inflight`
Total inflight (live) transactions.
- `isFrozen`
Is transaction system frozen (true or false)?
- `inflight-tx`
List of inflight transactions.

Use the following command to get Transaction Manager statistics:

```
asadmin get -m serverInstance.transaction-service.*
```

Here is a sample of the output:

```
***** Stats for JTS *****
total-tx-completed = 244283
total-tx-rolled-back = 2640
total-tx-inflight = 702
isFrozen = False
inflight-tx =
Transaction Id , Status, ElapsedTime(msec)
000000000003C95A_00, Active, 999
```

Tuning the Transaction Manager

A distributed transactional system writes transactional activity into transaction logs so that they can be recovered later. But writing transactional logs will have some performance penalty. This property can be used to disable the transaction logging, where the performance is of utmost importance more than the recovery. This property, by default, won't exist in the `server.xml`.

The `automatic-recovery` and `key-point-interval` attributes have an impact on performance when using the transaction manager. When `automatic-recovery` is set to `true`, `disable-distributed-transaction-logging` will not be considered and transaction logging will always happen. If `automatic-recovery` is set to `false`, `disable-distributed-transaction-logging` will be considered to determine whether to write transaction logs or not.

automatic-recovery

This value, together with `disable-distributed-transaction-logging` attribute, has some impact on performance. Here is how it works.

1. If `automatic-recovery` is `true`, transaction logs will always be written.
2. If `automatic-recovery` is `false` and `disable-distributed-transaction-logging` is off (default case), then the logs will be written.
3. If `automatic-recovery` is `false` and `disable-distributed-transaction-logging` is on, then the transaction logs will not be written. This will give approximately 20% improvement in performance but at the cost of not recovering as there won't be any transaction logs. In other words, transaction logging in case 1 and 2 results in approximately 20% impact. All these results apply only to global transactions intensive tests. Gains in real applications may be less.

keypoint-interval

The default value of this attribute is 2048. Key pointing prevents the physical log for a process from growing indefinitely by defining the frequency at which the log file may be cleaned up by removing entries for completed transactions. Frequent checkpointing is detrimental for performance. In most of the cases, the default value is good enough.

References

- For details on profiling, see the *Application Server Developer's Guide*, chapter on *Developing J2EE Applications*; specifically the section titled *Profiling Tools*.
- For more details on SNMP monitoring see the *Application Server Administrator's Guide*, the chapter on *Monitoring and Managing Applications*.

- For more details on the `server.xml` file see the *Application Server Configuration File Reference*.

Related Considerations

Tuning the Java Runtime System

The Solaris operating environment, by default, supports a two level thread model (up to Solaris 8). Application level Java threads are mapped to user level Solaris threads, which are multiplexed on a limited pool of light weight processes (LWPS). Often, we need only as many LWPS as there are processors on the system, leading to conserved kernel resources and greater system efficiency. This helps when there are hundreds of user level threads. Fortunately (or unfortunately), you can choose from multiple threading models and different methods of synchronization within the model, but this varies from VM to VM. Adding to the confusion, the threads library will be transitioning from Solaris 8 to 9, eliminating many of the choices. Although we have a 2 level model, in the 1.4 VM, we have an effectively 1-to-1 thread/lwp model since the VM used LWP based sync by default.

The following topics are discussed in this module:

- [Using Alternate Threads](#)
- [Managing Memory and Allocation](#)

Using Alternate Threads

You can try to load the alternate `libthread.so` in `/usr/lib/lwp/` on Solaris 8 by changing your `LD_LIBRARY_PATH` to include `/usr/lib/lwp` before `/usr/lib`. Both give better throughput and system utilization for certain applications; especially those using fewer threads.

By default, the Application Server uses `/usr/lib/lwp`. You can change the default settings to not use the LWP by removing `/usr/lib/lwp` from the `LD_LIBRARY_PATH` in the `startserv` script, but should be avoided unless required.

For applications using many threads, `/usr/lib/libthread.so` is the best library to use. Of course, see using `-Xconcurrentio` for applications with many threads as this will not only turn on LWP based sync, the default in 1.4, but also turn off TLABS, or thread local allocation buffers, which can chew up the heap and cause premature gcs.

To further examine the threading issues on Solaris with Java, see <http://java.sun.com/docs/hotspot/threads/threads.html>

For additional information on tuning the HotSpot JVM, see “HotSpot Virtual Machine Tuning Options” on page 112.

Managing Memory and Allocation

The efficient running of any application depends on how well memory and garbage collection are managed. The following sections provide information on optimizing memory and allocation functions:

- [Tuning the Garbage Collector](#)
- [Tracing Garbage Collection](#)
- [Specifying Other Garbage Collector Settings](#)
- [Tuning the Java Heap](#)
- [HotSpot Virtual Machine Tuning Options](#)

Tuning the Garbage Collector

Garbage collection reclaims the heap space previously allocated to objects no longer needed. The process of locating and removing the dead objects can stall any application while consuming as much as 25 percent throughput.

Almost all Java Runtime Environments come with a generational object memory system and sophisticated garbage collection algorithms. A generational memory system divides the heap into a few carefully sized partitions called generations. As these objects accumulate a low memory condition occurs forcing garbage collection to take place. The efficiency of a generational memory system is based on the observation that most of the objects are short lived. The heap space is divided into the old and the new generation.

The new generation includes the new object space (Eden), and two survivor spaces. New objects allocate in eden. Longer lived objects are moved from the new generation and tenured to the old generation.

The young generation uses a fast copying garbage collector which employs two semi-spaces (survivor spaces) in the eden, copying surviving objects from one survivor space to the second. Objects that survive multiple young space collections are tenured -- copied to a tenured generation. The tenured generation is larger and fills up less quickly. So, it is garbage collected less frequently; and each collection takes longer than a young space only collection. Collecting the tenured space is also referred to as doing a full Generation Collection (GC).

The frequent young space collections are quick (few milliseconds), and the occasional full GC takes a relatively longer time (tens of milliseconds to even a few seconds, depending upon the heap size).

Other garbage collection algorithms, such as the Train algorithm, are incremental. They chop down the full GC into several incremental pieces. This provides a high probability of small garbage collection pauses even when full GC takes affect. This comes with an overhead and is not required for enterprise web applications.

When the new generation fills up, it triggers a minor collection in which the surviving objects are moved to the old generation. When the old generation fills up, it triggers a major collection which involves the entire object heap.

Both HotSpot and Solaris JDK use thread local object allocation pools for lock-free, fast, and scalable object allocation. User application level object pooling may have been more beneficial running on earlier generation Java Virtual Machines. Consider pooling only if the object construction cost is very high and shows up being significant in the execution profiles.

Choosing the Garbage Collection Algorithm

Long pauses during a Full Garbage Collection spanning more than 4 seconds can produce intermittent failures in persisting Session data into HADB.

While garbage collection is going on, the Application Server isn't running. If the pause is long enough, the HADB times out the existing connections. Then, when the application server resumes its activities, the HADB generates errors when the application server attempts to use those connections to persist session data (errors like, "Failed to store session data", "Transaction Aborted", or "Failed to connect to HADB server".)

To prevent that problem, use the CMS collector as the garbage collection algorithm. This Collector may cause a drop in thruput for heavily utilized systems, because it is running more or less constantly. But it prevents the long pauses that can occur when the garbage collector runs infrequently.

To use the CMS collector:

1. Make sure that the system isn;t maxed out with respect to CPU utlitzation.
2. Configure HADB time outs, as deccribed in the *System Deployment Guide*, Appendix X.
3. Configure the CMS collector in the server instance by adding the following entry to `server.xml`:

```
<jvm-options>
-XX:+UseConcMarkSweepGC -XX:SoftRefLRUPolicyMSPerMB=1
</jvm-options>
```

Additional Information

Use the `jvmsstat` utility to monitor Hotspot garbage collection. (See “[HotSpot Virtual Machine Tuning Options](#)” on page 112.)

For detailed information on tuning the garbage collector, see <http://java.sun.com/docs/hotspot/gc1.4.2/index.html>

Tracing Garbage Collection

Two primary measures of garbage collection performance are Throughput and Pauses. Throughput is the percentage of the total time spent on other activities apart from garbage collection.

Pauses are times when an application appears unresponsive due to garbage collection. Users can have different requirements of garbage collection. A server centric application may consider throughput to be a metric, but a short pause may upset a graphical program. There are two other considerations - Footprint and promptness.

Footprint

Footprint is the working set of a process, measured in pages and cache lines. Promptness is the time between when an object becomes dead, and when the memory becomes available. This is an important consideration for distributed systems.

A particular generation sizing chooses a trade-off between these considerations. A very large young generation may maximize the throughput, but does so at the expense of footprint and promptness. Pauses can be minimized by using a small young generation and incremental collection.

Pauses due to Generation Collection are inspected by diagnostic output of the Java Virtual Machine. The command line argument `"-verbose:gc"` prints information at every collection. Below is a sample output of the information generated when this flag is passed to the Java Virtual Machine.

```
[GC 50650K->21808K(76868K), 0.0478645 secs]
[GC 51197K->22305K(76868K), 0.0478645 secs]
[GC 52293K->23867K(76868K), 0.0478645 secs]
[Full GC 52970K->1690K(76868K), 0.54789968 secs]
```

The numbers before and after the arrows indicate the combined size of live objects before and after the collection. The number in the parenthesis is the total available space, which is the total heap minus one of the survivor spaces. In this sample, there are three minor collections and one major collection. In the first GC, 50650 KB of objects existed before collection and 21808 KB of objects after collection. This means that 28842 KB of objects were dead and collected. The total heap size is 76868 KB. The collection process required 0.0478645 seconds.

Other useful monitoring options include:

- `-XX:+PrintGCDetails` for more detailed logging information
- `-Xloggc:file` to save the information in a log file

Specifying Other Garbage Collector Settings

For applications which do not dynamically generate and load classes, permanent generation is not relevant to the GC performance. For applications which dynamically generate and load classes (JSP's), the permanent generation is relevant to the GC performance, as filling the permanent generation can trigger a Full GC. The maximum permanent generation can be tuned with `-XX:MaxPermSize` option.

Applications can interact with the garbage collection by invoking collections explicitly through the `System.gc()` call. But, relying on the application to manage the resources is a bad idea since these force major collections, and inhibit scalability on large systems. This can be disabled by using the flag `-XX:+DisableExplicitGC`.

The Application Server uses RMI in the Administration module for monitoring. Garbage cannot be collected in RMI based distributed applications without occasional local collections, so RMI forces a periodic full collection. The frequency of these collections can be controlled with the property

`-sun.rmi.dgc.client.gcInterval`. For example, `- java -Dsun.rmi.dgc.client.gcInterval=3600000` specifies explicit collection once per hour instead of the default rate of once per minute.

To specify the attributes for the Java virtual machine:

- Edit the `server.xml` by hand and manually add `<jvm-options>vm tunable</jvm-options>` where `vm tunable` is the tuning that the user wants to apply.
- Using the administration interface and JVM settings | JVM options.

Tuning the Java Heap

This section discusses topics related to tuning the Java Heap for performance.

- [Guidelines for Java Heap Sizing](#)
- [Heap Tuning Parameters](#)
- [Sample Heap Configuration on Solaris](#)

Guidelines for Java Heap Sizing

Maximum heap size depends on maximum address space per process. Here are the maximum per-process address values for the various platforms:

x86 / Redhat Linux 32 bit	2 GB
x86 / Redhat Linux 64 bit	3 GB
x86 / Win98/2000/NT/Me/XP	2 GB
x86 / Solaris x86 (32 bit)	4 GB
Sparc / Solaris 32 bit	4 GB
Sparc / Solaris 64 bit	terabytes

Of course, the maximum heap space is always smaller than maximum address space per process, because the process also needs space for stack, libraries, and so on. To determine the maximum heap space that can be allocated, use a profiling tool to examine the way memory is used. Gauge the maximum stack space the process uses and the amount of memory taken up libraries and other memory structures. The difference between the maximum address space and the total of those values is the amount of memory that can be allocated to the heap.

Heap Tuning Parameters

You control the heap size using a variety of parameters.

The `-Xms` and `-Xmx` parameters defines the minimum and maximum heap sizes. Since collections occur when the generations fill up, throughput is inversely proportional to the amount of the memory available. By default, the JVM grows or shrinks the heap at each collection to try to keep the proportion of free space to the living objects at each collection within a specific range. This range is set as a percentage by the parameters `-XX:MinHeapFreeRatio=<minimum>` and `-XX:MaxHeapFreeRatio=<maximum>`; and the total size bounded by `-Xms` and `-Xmx`.

Server side applications set the values of `-Xms` and `-Xmx` equal to each other for a fixed heap size. When the heap grows or shrinks, the JVM must recalculate the old and new generation sizes to maintain a predefined `NewRatio`.

The `NewSize` and `MaxNewSize` parameters control the new generation's minimum and maximum size. You can regulate the new generation size by setting these parameters equal. The bigger the younger generation, the less often minor collections occur. By default, the young generation is controlled by `NewRatio`. For example, setting `-XX:NewRatio=3` means that the ratio between the old and young generation is 1:3, the combined size of eden and the survivor spaces will be fourth of the heap. In order to be safe, set the `NewSize/MaxNewSize` to the same value.

By default, the Application Server is invoked with the Java HotSpot Server JVM. The default `NewRatio` for the Server JVM is 2, the old generation occupies 2/3 of the heap while the new generation occupies 1/3. The larger new generation can accommodate many more short lived objects, decreasing the need for slow major collections. The old generation is still sufficiently large enough to hold many long-lived objects.

The following are important guidelines for sizing Java heap.

- Decide the total amount of memory you can afford to the JVM. Accordingly, graph your own performance metric against young generation sizes to find the best setting.
- Make plenty of memory available to the young generation, the default for 1.4 is calculated from `NewRatio` and the `-Xmx` setting.
- Larger Eden or younger generation spaces increase the spacing between full garbage collections. But young space collections could take a proportionally longer time. In general, you could keep the eden size between 1/4th and 1/3rd the maximum heap size.

Old generation must be typically larger than the new generation.

Survivor Ratio Sizing

The `SurvivorRatio` parameter controls the size of the two survivor spaces. For example, `-XX:SurvivorRatio=6` sets the ratio between each survivor space and eden to be 1:6, each survivor space will be one eighth of the young generation. With JDK 1.4, the Solaris default is 32. If survivor spaces are too small, copying collection overflows directly into the old generation. If survivor spaces are too large, they will be empty. At each garbage collection, the JVM chooses a threshold number of times an object can be copied before it is tenured.

This threshold is chosen to keep the survivors half full.

The option `-XX:+PrintTenuringDistribution` can be used to show the threshold and ages of the objects in the new generation. It is useful for observing the lifetime distribution of an application.

For up-to-date defaults, refer to <http://java.sun.com/docs/hotspot/VMOptions.html>

Sample Heap Configuration on Solaris

This is a sample heap configuration used by Application Server for heavy server-centric applications, on Solaris, as set in the `server.xml` file.

```
<jvm-options> -Xms3584m </jvm-options>
<jvm-options> -Xmx3584m </jvm-options>
<jvm-options> -verbose:gc </jvm-options>
<jvm-options> -Dsun.rmi.dgc.client.gcInterval=3600000 </jvm-options>
```

HotSpot Virtual Machine Tuning Options

HotSpot is a just-in-time byte-code compiler for Java applications that interprets seldom-used code to avoid the overhead of optimization. It has excellent performance characteristics straight out of the box, but it also has a variety of tuning options that can be customized to maximize performance of the Application Server.

Start by using the `jvmsstat` utility to monitor HotSpot, so you can see where improvements are needed and see the effect of changes on performance:

- CoolStuff—`jvmsstat`
<http://developers.sun.com/dev/coolstuff/jvmsstat/v2/docs.html>

Then consult the following guides to tune for maximum results:

- Java HotSpot VM Options <http://java.sun.com/docs/hotspot/VMOptions.html>

- **Frequently Asked Questions About the Java HotSpot Virtual Machine**
<http://java.sun.com/docs/hotspot/PerformanceFAQ.html>
- **Additional documents on HotSpot tuning**
<http://java.sun.com/docs/hotspot/>
- **The main java performance web page:**
<http://java.sun.com/performance/>

Tuning the Operating System

Tuning Solaris TCP/IP settings benefits programs that open and close many sockets. The Application Server operates with a small fixed set of connections and the performance gain may not be as significant on the Application Server node. Improvements for the Web Server, configured as a Web front-end to Application Server, can have significant benefits. The following topics are discussed:

- [Tuning Parameters](#)
- [Solaris File Descriptor Setting](#)
- [General Performance tuning for solaris on x86](#)
- [Tuning for Linux platforms](#)

Tuning Parameters

The following table shows operating system tuning, for Solaris, used when benchmarking for performance and scalability. These values are an example of how you may tune your system to achieve the desired result.

Table 5-1 Tuning the Solaris Operating System

Parameter	Scope	Default Value	Tuned Value	Comments
rlim_fd_max	/etc/system	1024	8192	Process open file descriptors limit; should account for the expected load (for the associated sockets, files, pipes if any).
rlim_fd_cur	/etc/system	1024	8192	

Table 5-1 Tuning the Solaris Operating System

Parameter	Scope	Default Value	Tuned Value	Comments
sq_max_size	/etc/system	2	0	Controls streams driver queue size; setting to 0 makes it infinity so the performance runs wont be hit by lack of buffer space. Set on clients too.
tcp_close_wait_interval	ndd /dev/tcp	240000	60000	Set on clients as well.
tcp_time_wait_interval	ndd /dev/tcp	240000	60000	
tcp_conn_req_max_q	ndd /dev/tcp	128	1024	
tcp_conn_req_max_q0	ndd /dev/tcp	1024	4096	
tcp_ip_abort_interval	ndd /dev/tcp	480000	60000	
tcp_keepalive_interval	ndd /dev/tcp	7200000	900000	For high traffic web sites lower this value.
tcp_rexmit_interval_initial	ndd /dev/tcp	3000	3000	If retransmission is greater than 30-40%, you should increase this value.
tcp_rexmit_interval_max	ndd /dev/tcp	240000	10000	
tcp_rexmit_interval_min	ndd /dev/tcp	200	3000	
tcp_smallest_anon_port	ndd /dev/tcp	32768	1024	Set on clients too.
tcp_slow_start_initial	ndd /dev/tcp	1	2	Slightly faster transmission of small amounts of data.
tcp_xmit_hiwat	ndd /dev/tcp	8129	32768	To increase the transmit buffer.
tcp_rcv_hiwat	ndd /dev/tcp	8129	32768	To increase the transmit buffer.
tcp_rcv_hiwat	ndd /dev/tcp	8129	32768	To increase the receive buffer.

Table 5-1 Tuning the Solaris Operating System

Parameter	Scope	Default Value	Tuned Value	Comments
tcp_conn_hash_size	ndd /dev/tcp	512	8192	The connection hash table keeps all the information for active TCP connections (ndd -get /dev/tcp tcp_conn_hash). This value does not limit the number of connections, but it can cause connection hashing to take longer. To make lookups more efficient, set the value to half of the number of concurrent TCP connections that you expect on the server (netstat -nP tcp wc -l, gives you a number). It defaults to 512. This can only be set in /etc/system and becomes effective at boot time.

Solaris File Descriptor Setting

On Solaris, setting the maximum number of open files property using `ulimit` has the biggest impact on your efforts to support the maximum number of RMI/IIOP clients.

To increase the hard limit, add the following command to `/etc/system` and reboot it once:

```
set rlim_fd_max = 8192
```

You can verify this hard limit by using the following command:

```
ulimit -a -H
```

Once the above hard limit is set, you can increase the value of this property explicitly (up to this limit) using the following command:

```
ulimit -n 8192
```

You can verify this limit by using the following command:

```
ulimit -a
```

For example, with the default `ulimit` of 64, a simple test driver can support only 25 concurrent clients, but with `ulimit` set to 8192, the same test driver can support 120 concurrent clients. The test driver spawned multiple threads, each of which performed a JNDI lookup and repeatedly called the same business method with a think (delay) time of 500ms between business method calls, exchanging data of about 100KB.

These settings apply to RMI/IIOP clients (on Solaris). Refer to Solaris documentation on the Sun Microsystems documentation web site (www.docs.sun.com) for more information on setting the file descriptor limits.

For further information on solaris system tunables, please refer to the Solaris Tunable Parameters reference manual located at <http://docs.sun.com/db/doc/806-7009>.

General Performance tuning for solaris on x86

The following are some options that need to be considered when tuning solaris on x86 platform for the application server and HADB. Please note that some of the values will depend on the system resources available.

The following should be added to the `/etc/system`. These affect the number of semaphores and also the shared memory settings. These are more relevant for the machine on which HADB server is running.

These settings affect the shared memory and semaphores on the system:

```
set shmsys:shminfo_shmmax=0xffffffff
set shmsys:shminfo_shmseg=128
set semsys:seminfo_semmnu=1024
set semsys:seminfo_semmap=128
set semsys:seminfo_semmni=400
set semsys:seminfo_semmns=1024
```

These settings are for the file descriptors:

```

set rlim_fd_max=65536
set rlim_fd_cur=65536
set sq_max_size=0
set tcp:tcp_conn_hash_size=8192
set autoup=60
set pcisch:pci_stream_buf_enable=0

```

These settings are for tuning the IP stack:

```

set ip:tcp_squeue_wput=1
set ip:tcp_squeue_close=1
set ip:ip_squeue_bind=1
set ip:ip_squeue_worker_wait=10
set ip:ip_squeue_profile=0

```

Place the following changes to the default tcp variables in a startup script that gets executed when the system reboots, or else you will lose your changes:

```

nnd -set /dev/tcp tcp_time_wait_interval 60000
nnd -set /dev/tcp tcp_conn_req_max_q 16384
nnd -set /dev/tcp tcp_conn_req_max_q0 16384
nnd -set /dev/tcp tcp_ip_abort_interval 60000
nnd -set /dev/tcp tcp_keepalive_interval 7200000
nnd -set /dev/tcp tcp_rexmit_interval_initial 4000
nnd -set /dev/tcp tcp_rexmit_interval_min 3000
nnd -set /dev/tcp tcp_rexmit_interval_max 10000
nnd -set /dev/tcp tcp_smallest_anon_port 32768
nnd -set /dev/tcp tcp_slow_start_initial 2
nnd -set /dev/tcp tcp_xmit_hiwat 32768
nnd -set /dev/tcp tcp_recv_hiwat 32768

```

Note:

After making any changes to the `/etc/system`, you need to reboot the machines.

Tuning for Linux platforms

To tune for maximum performance on Linux, you'll:

- [Increase the number of file descriptors](#)
- [Change the virtual memory settings](#)
- [Ensure that the Network interface is operating in full duplex mode](#)
- [Tune disk I/O performance](#)
- [Tune the TCP/IP stack](#)

Increase the number of file descriptors

Having more number of file desc. ensures that the server can open the sockets under highload and thus not abort requests coming in from the clients.

Start by checking system limits for file descriptors with this command:

```
% cat /proc/sys/fs/file-max
8192
```

The current limit shown is 8192. To increase it to 65535 (as root):

```
# echo "65535" > /proc/sys/fs/file-max
```

If you want this new value to survive across reboots you can add it to `/etc/sysctl.conf` o specify the maximum number of open files permitted:

```
fs.file-max = 65535
```

Note: The parameter is *not* `proc.sys.fs.file-max`, as one might expect.

To list the available parameters that can be modified using `sysctl`:

```
% sysctl -a
```

To load new values from the `sysctl.conf` file:

```
% sysctl -p /etc/sysctl.conf
```

To check and modify limits per shell:

```
% limit
cputime      unlimited
filesize     unlimited
datasize     unlimited
stacksize    8192 kbytes
coredumpsize 0 kbytes
memoryuse     unlimited
descriptors  1024
memorylocked  unlimited
maxproc      8146
openfiles    1024
```

The `openfiles` and `descriptors` show a limit of 1024. To increase the limit to 65535 for all users, edit `/etc/security/limits.conf` as root, and modify or add the "nofile" (number of file) entries:

```
*      soft    nofile    65535
*      hard    nofile    65535
```


(Here “*” is a wildcard that identifies all users. You could also specify a userid, instead.)

Then edit `/etc/pam.d/login` and add the line:

```
session    required    /lib/security/pam_limits.so
```

Note:

On RedHat, you also need to edit `/etc/pam.d/sshd` and add the line,
`session required /lib/security/pam_limits.so`

On many systems, this procedure will be sufficient. Log in as a regular user and try it before doing the remain steps. (They may not be required, depending on how PAM and ssh are configured.)

Change the virtual memory settings

Add the following to `/etc/rc.local`

```
echo 100 1200 128 512 15 5000 500 1884 2 > /proc/sys/vm/bdflush
```

For more information view the manpages for `bdflush`.

For HADB settings please refer to [Chapter 6, “Tuning for High-Availability”](#).

Ensure that the Network interface is operating in full duplex mode

Add the following entry into `/etc/rc.local`

```
mii-tool -F 100baseTx-FD eth0, where eth0 is the name of the NIC.
```

Tune disk I/O performance

For non scsi disks:

1. Test the speed using

```
/sbin/hdparm -t /dev/hdX
```

2. Enable DMA:

```
/sbin/hdparm -d1 /dev/hdX
```

3. Check the speed again using the `hdparm` command.1

Given that DMA wasn't enabled by default, the transfer rate should now have improved considerably. In order to do this at every reboot, add the `/sbin/hdparm -d1 /dev/hdX` line to your `/etc/conf.d/local.start`, `/etc/init.d/rc.local`, or whatever you call your startup script.

For SCSI Disks: Please refer to the following URL:

http://people.redhat.com/alikins/system_tuning.html#scsi

Tune the TCP/IP stack

Add the following entry below into the `/etc/rc.local`

```
echo 30 > /proc/sys/net/ipv4/tcp_fin_timeout
echo 60000 > /proc/sys/net/ipv4/tcp_keepalive_time
echo 15000 > /proc/sys/net/ipv4/tcp_keepalive_intvl
echo 0 > /proc/sys/net/ipv4/tcp_window_scaling
```

And add the following to `/etc/sysctl.conf`

```
# Disables packet forwarding
net.ipv4.ip_forward = 0

# Enables source route verification
net.ipv4.conf.default.rp_filter = 1

# Disables the magic-sysrq key
kernel.sysrq = 0

net.ipv4.ip_local_port_range = 1204 65000
net.core.rmem_max = 262140
net.core.rmem_default = 262140
net.ipv4.tcp_rmem = 4096 131072 262140
net.ipv4.tcp_wmem = 4096 131072 262140
net.ipv4.tcp_sack = 0
net.ipv4.tcp_timestamps = 0
net.ipv4.tcp_window_scaling = 0
net.ipv4.tcp_keepalive_time = 60000
net.ipv4.tcp_keepalive_intvl = 15000
net.ipv4.tcp_fin_timeout = 30
```

Then, add the following as the last entry in `/etc/rc.local`

```
sysctl -p /etc/sysctl.conf
```

Note:

After making these changes, you need to reboot the system.

Finally, use this command to increase the size of the transmit buffer:

```
tcp_recv_hiwat ndd /dev/tcp 8129 32768
```


Tuning for High-Availability

This chapter discusses the following topics:

- [Tuning HADB](#)
- [Tuning the Application Server for High-Availability](#)

Tuning HADB

The high-availability database (HADB) used for storing persistent session state must be tuned according to the load of the Application Server. The data volume, transaction frequency, and size of each transaction may affect the performance of the HADB, and consequently the performance of Application Server.

The following topics are discussed in this chapter:

- [Disk Usage](#)
- [Memory Allocation](#)
- [Performance](#)
- [Operating System Configuration](#)

Disk Usage

This section discusses how to calculate HADB data device size and explains the use of separate disks for multiple data devices.

Calculating HADB Data Device Size

When the HADB database is created, you need to specify the number, and size of each data device. These devices must have room for all the user data to be stored. In addition, extra space must be allocated to account for internal overhead as discussed in the following section.

If the database runs out of device space, the HADB returns error codes 4593 or 4592 to the Application Server.

NOTE	See the <i>Application Server Error Message Reference</i> for more information on these error messages.
-------------	---------------------------------------------------------------------------------------------------------

These error messages are also written to the history file(s). In this case, the HADB blocks any client requests to insert, or update data. However, Delete operations are accepted.

In the Application Server context, the HADB is used for storing session states as binary data. The session state is serialized, and stored as a BLOB (binary large object). Each BLOB is split into chunks of approximately, 7KB each. Each chunk is stored as a database row (context row is synonymous with tuple, or record). Database rows are stored in pages, of size 16KB.

There is a small internal overhead for each row stored (approximately 30 bytes for the Application Server). With the most compact allocation of rows (BLOB chunks) two rows can be stored in a page. Internal fragmentation in the B-tree storage structure may result in only one row per page. On average, 50% of each page contains user data.

For availability in case of node failure, HADB always replicates user data. The HADB node stores its own data, plus a copy of the data from its mirror node. Hence, all data is stored twice. Since 50% of the space on a node is user data (on average), and each node is mirrored, the data devices must have space for at least four times the volume of the user data.

In the case of data refragmentation, HADB keeps both the old and the new versions of a table while the refragmentation operation is running. All application requests are performed on the old table, while the new table is being created. Assuming that the database is primarily used for one huge table containing BLOB data for session states, this means we must multiply the device space requirement with another factor of two. Consequently, if we want to be able to add nodes to a running database, and refragment the data to use all nodes, we actually need eight times the volume of user data available.

In addition to the factors described above, you must also account for the device space that the HADB reserves for its internal use. (Four times that of the `LogBufferSize`). This space on the disk is used for temporary storage of the log buffer (see “[hadbm deviceinfo --details](#)” on page 129) during high load conditions.

Tuning DataDeviceSize

To increase the size of the data device(s) of HADB, use the following command:

```
hadbm set TotalDataDeviceSizePerNode
```

The `hadbm` restarts all the nodes, one by one, for the change to take effect. For more information on using this command, see the *Application Server Administrator's Guide*, “Configuring the High-Availability Database” chapter.

NOTE The current version of `hadbm` does not add data devices to a running database instance.

See the *Application Server System Deployment Guide* for detailed information.

Placing HADB files on Physical Disks

For best performance, data devices should be allocated on separate physical disks. This applies if there are nodes with more than one data device, or if there are multiple nodes on the same host.

Placing devices belonging to different nodes on different devices is strongly recommended, in general. But it is especially recommended for Red Hat AS 2.1, because the HADB nodes have been observed to wait for asynchronous I/O when the same disk is used for devices that belong to more than one node.

An HADB node writes information, warnings and errors to the history file synchronously, rather than using the asynchronous writes normally associated with output devices. Therefore, any disk waits when writing to the history file affect HADB behavior and performance. This situation is indicated by the following message in the history file:

```
BEWARE - last flush/fputs took too long
```

To avoid this problem, keep the HADB executables and the history files on physical disks other than those keeping the data devices.

Memory Allocation

It is essential to ensure that HADB is allocated sufficient memory, especially when HADB is co-located with other processes.

The HADB Node Supervisor Process (`NSUP`) tracks the time elapsed since the last time it did some monitoring work. If that time duration exceeds a specified maximum (2500 ms, by default), `NSUP` concludes that it was blocked too long and restarts the node. The situation is especially likely arise when there are other processes in the system that compete for memory, which produces extensive swapping and multiple page faults as processes are rescheduled.

Then, when the blocked node restarts, all active transactions on that node are aborted.

If you see that the Application Server throughput slows down and requests abort or time out, make sure that swapping is not the cause. To monitor swapping activity, use this command on Unix systems:

```
vmstat -S
```

In addition, look for this message in the HADB history files. It is written when the HADB node is restarted, where `M` is greater than `N`:

```
Process blocked for .M. sec, max block time is .N. sec
```

The presence of aborted transactions will be signaled by the error message

```
HADB00224: Transaction timed out or HADB00208: Transaction aborted.
```

Performance

For best performance, all HADB processes (`clu_XXX_srv`) should fit in the physical memory. They should not be paged or swapped. The same applies for shared memory segments in use. You can configure the size of some of the shared memory segments. If these segments are too small, performance suffers – user transactions are delayed, or even aborted. If the segments are too large then the physical memory is wasted.

The following parameters may be configured:

- [DataBufferPoolSize](#)
- [LogbufferSize](#)
- [InternalLogbufferSize](#)
- [NumberOfLocks](#)

- [Timeouts](#)

DataBufferPoolSize

HADB stores data on data devices, which are allocated on disks. The data must be in the main memory before it can be processed. The HADB node allocates a portion of shared memory for this purpose. If the allocated database buffer is small compared to the data being processed, a lot of processing capacity is wasted on the disk input/output. In a system with write-intensive operations (for example, session states that are frequently updated) the database buffer must be big enough so that the processing capacity used on the disk input/output does not hamper the request processing.

The database buffer is similar to a cache in a file system. For good performance the cache must be utilized as much as possible, so that there is no need to wait for a disk read operation. For best performance, the entire database contents should fit in the database buffer. In most settings, this is not feasible. You must aim to have the “working set” of the client applications in the buffer. Disk input/output should also be monitored. If HADB performs many disk read operations, this means that the database is low on buffer space. The database buffer is partitioned into blocks of size 16KB. The same block size is used on the disk. HADB schedules multiple blocks for read/write in one input/output operation. HADBM can be used to monitor disk usage:

Table 6-1 hadbm deviceinfo --details

Node No.	TotalSize	Free Size	Usage
0	512	504	1%
1	512	504	1%

Table 6-2 hadbm resourceinfo --databuf

Node No.	Avail	Free	Access	Misses	Copy-on-Write
0	32	0	205910260	8342738	400330
1	32	0	218908192	8642222	403466

Legend

TotalSize	Size of device, in megabytes
FreeSize	Free size, in megabytes

Usage	Percent used.
Avail	Size of buffer, in megabytes.
Free	Free size, in megabytes, when the data volume is larger than the buffer. (The entire buffer is used at all times.)
Access	Number of times blocks that have been accessed in the buffer.
Misses	Number of block requests that "missed the cache" that is, the user had to wait for a disk read.
Copy-on-Write	Number of times the block has been modified while it is being written to disk.

For a well tuned system, the number of misses (and hence the number of reads) must be very small compared to the number of writes. The example numbers above show a miss rate of about 4% (200 million access, and 8 million misses). The acceptability of these figures depends on the client application requirements.

Tuning DataBufferPoolSize

To change the size of the database buffer, use the following command:

```
hadbm set DataBufferPoolSize
```

The `hadbm` restarts all the nodes, one by one, for the change to take effect. For more information on using this command, see the *Application Server Administrator's Guide*, "Configuring the High-Availability Database" chapter.

LogbufferSize

User requests consist of operations such as inserting, deleting, updating or reading data. The HADB logs all operations which modify the database before executing them. Log records describing the operations are placed in a portion of shared memory, referred to as the (tuple) log buffer. These log records are used for undoing operations in case transactions are aborted, for recovery in case of node crash, and for replication between mirror nodes.

The log records remain in the buffer until they are processed locally, and shipped to the mirror node. The log records are kept until the outcome (commit or abort) of the transaction is certain. If the HADB node runs low on tuple log, the user transactions are delayed, and possibly timed out.

Tuning the Attribute

Begin with the default value. Look for `HIGH LOAD` informational messages in the history files. All the relevant messages will contain `tuple log` or simply `log`, and a description of the internal resource contention that occurred.

The following command gives information on log buffer size and usage:

```
hadbm resourceinfo --logbuf
```

Under normal operation the log is reported as 70-80% full. This is because space reclamation is said to be "lazy". HADB requires as much data in the log as possible, in case there is a node crash and recovery needs to be performed.

Table 6-3 `hadbm resourceinfo --logbuf`

Node No.	Avail	Free Size
0	44	42
1	44	42

Legend

Node No.	The node number.
Avail	Size of buffer, in megabytes.
Free Size	Free size, in megabytes when the data volume is larger than the buffer. The entire buffer is used at all times.

To change the size of the log buffer use the following command:

```
hadbm set LogbufferSize
```

The `hadbm` restarts all the nodes one by one, for the change to take effect. For more information on using this command, see the *Application Server Administrator's Guide*, "Configuring the High-Availability Database" chapter.

InternalLogbufferSize

The node internal log (`nilog`) contains information about physical (as opposed to logical, row level) operations at the local node. For example, it provides information on whether there are disk block allocations and deallocations, and B-tree block splits. This buffer is maintained in shared memory, and is also checked to disk (a separate log device) at regular intervals. The page size of this buffer, and the associated data device is 4096 bytes.

Large BLOBs necessarily allocate many disk blocks, and thus create a high load on the node internal log. This is normally not a problem, since each entry in the `nilog` is small.

Tuning the Attribute

Begin with the default value. Look out for `HIGH LOAD` informational messages in the history files. The relevant messages will contain `nilog`, and a description of the internal resource contention that occurred.

Table 6-4 `hadbm resourceinfo --nilogbuf`

Node No.	Avail	Free Size
0	11	11
1	11	11

Legend

- Node No.** The node number.
- Avail** Size of buffer, in megabytes.
- Free Size** Free size, in megabytes when the data volume is larger than the buffer.
The entire buffer will be used at all times.

To change the size of the `nilog` buffer, use the following command:

```
hadbm set InternalLogbufferSize
```

The `hadbm` restarts all the nodes, one by one, for the change to take effect. For more information on using this command, see the *Application Server Administrator's Guide*, “Configuring the High-Availability Database” chapter.

NOTE

If you change the size of the `nilog` buffer, the associated log device (located in the same directory as the data devices) also changes. The size of the internal log buffer must be equal to the size of the internal log device The command `hadbm set InternalLogBufferSize` ensures this requirement. It stops a node, increases the `InternalLogBufferSize`, re initializes the internal log device, and brings up the node. This sequence is performed on all nodes.

NumberOfLocks

Each row level operation requires a lock in the database. Locks are held until transaction commits or rolls back. Locks are set at the row (BLOB chunk) level, which means that a large session state requires many locks. Locks are needed for both primary, and mirror node operations. Hence, a BLOB operation allocates the same number of locks on two HADB nodes.

When a table refragmentation is performed, HADB needs extra lock resources. Thus, ordinary user transactions can only acquire half of the locks allocated.

If the HADB node has no lock objects available, errors are written to the log file. See error codes `HADB02080` and `HADB02096` in the *Application Server Error Message Reference* for a description of the error message and action to be taken to correct the error.

To calculate the number of locks

To calculate the number of locks you need to estimate the following parameters:

- Number of concurrent users that request session data to be stored in HADB (one session record per user)
- Maximum size of the BLOB session
- Persistence scope (max session data size in case of session/modified session and maximum number of attributes in case of modified session)

Assume that the maximum concurrent users at any time is x , i.e., x session data records are present in the HADB, and the session size (for session/modified session) or attribute size (for modified attribute) is y . The number of records written to HADB is as follows:

$$x * ((y/7000) + 2)$$

For record operations such as insert, delete, update and read, one lock will be used per record.

NOTE Locks are held for both primary records and hot-standby records. Hence, for insert, update and delete operations a transaction will need twice as many locks as the number of records. Read operations need locks only on the primary records. During refragmentation and creation of secondary indices, log records for the involved table are also sent to the fragment replicas being created. In that case, a transaction needs four times as many locks as the number of involved records. (Assuming all queries are for the affected table.)

Summary

Number of locks to be configured is as follows:

$nlocks = 4 * X * ((Y / 7000) + 2)$ This is if refragmentation is performed.

or

$nlocks = 2 * X * ((Y / 7000) + 2)$, otherwise.

Tuning the Attribute

Start with the default value. Look for exceptions with the indicated error codes in the Application Server log files. To get information on allocated locks and locks in use, use the following command:

```
hadbm resourceinfo --locks
```

Remember that under normal operations (no ongoing refragmentation) only half of the locks may be acquired by the client application.

Table 6-5 hadbm resourceinfo --locks

Node No.	Avail	Free	Waits
0	50000	50000	na
1	50000	50000	na

Legend

- Node No.** The node number.
- Avail** Number of locks available.
- Free** Number of locks in use.
- Waits** Number of transactions that have waited for a lock.
 "na" (not applicable) if all locks are available.

To change the number of locks you can use the following command:

```
hadbm set NumberOfLocks
```

The `hadbm` restarts all the nodes, one by one, for the change to take effect. For more information on using this command, see the *Application Server Administrator's Guide*, "Configuring the High-Availability Database" chapter.

Timeouts

This section describes some of the timeout values that affect performance.

TIP Timeouts are documented in the DTD files. In particular, see `lb.dtd` for load balancer timeouts, and `server.dtd` for server timeouts.

JDBC connection pool timeouts

These values govern how much time the server waits for a connection from the pool before it times out. In most cases, the default values work well. For detailed tuning information, see [“JDBC Connection Pool Tuning” on page 73](#).

Load Balancer timeouts

Some values to be aware of are:

- **response-timeout-in-seconds:** The time for which the load balancer plug-in (`lbplugin`) hosted in the webserver will wait for a response before it declares an instance dead and fails over to the next instance in the cluster. This value should be large enough to accommodate the maximum latency for a request from the server instance under the worst (high load) conditions.
- **health checker: interval-in-seconds:** Determines how frequently the load balancer pings the instance to see if it is healthy. Default value is 30 seconds. If the response-timeout-in-seconds is optimally tuned, and the server doesn't have too much traffic, then the default value works well.
- **health checker: timeout-in-seconds:** How long the load balancer waits after pinging an instance. Default value is 100 seconds, which is generally fine.

The combination of the health checker's interval-in-seconds and timeout-in-seconds values determine how much additional traffic goes from the load balancer plug-in (`lbplugin`) in the web server to the server instances.

HADB timeouts

Some HADB timeouts also make a difference.

- **sql_client time out:** This value is documented in the HADB guide.

Operating System Configuration

The following section describes configuration of the operating system.

Semaphores

If the number of semaphores is too low, HADB may fail and the following error message is displayed:

No space left on device

This may occur either while starting the database, or during run time. Since the semaphores are provided as a global resource by the operating system, the configuration depends on all processes running on the host, and not the HADB alone. In Solaris, you can configure the semaphore settings by editing the `/etc/system` file.

To run the nodes, `NNODES` (the number of nodes submitted implicitly by `--hosts` option to the HADB) and `NCONNS` connections (HADB configuration parameter `NumberOfSessions`, default value being 100) per host, the following semaphore settings may be used¹:

```
set semsys:seminfo_semmap = <default=10> + NNODES
set semsys:seminfo_semmni = <default=10> + NNODES
set semsys:seminfo_semmns = <default=60> + (NNODES * 8)
set semsys:seminfo_semmnu = <default=30> + NNODES + NCONNS
```

If you plan to run multiple nodes per host, make sure `semmap = NNODES`. The commands `sysinfo` and `sysdef` may be used to inspect the settings.

Shared Memory

Set the maximum shared memory size to the total amount of physical RAM. Additionally, the maximum number of shared memory segments per process should be set to at least six to accommodate the HADB processes. The number of system wide, shared memory identifiers, should be set depending on the number of nodes running on the host.

Solaris

In Solaris 9, because of the kernel changes, the `hmsys:shminfo_shmseg` variable is obsolete. In Solaris 8, add the following settings to the `/etc/system` file:

```
set shmsys:shminfo_shmmax = 0xffffffff
set shmsys:shminfo_shmseg = <default=6>
set shmsys:shminfo_shmmni = <default=100> + (6 * NNODES)
```

NOTE The host must be rebooted after changing these settings.

1. Default values are provided for Solaris 8. The HADB resource requirements should be added to the previous value of the variables regardless of whether these are the default values or not.

Linux

To increase the shared memory to 512 MB, run the following:

```
echo 536870912 > /proc/sys/kernel/shmmax
echo 536870912 > /proc/sys/kernel/shmall
```

Where the file `shmmax` contains the maximum size of a single shared memory segment, and `shmhall` contains the total shared memory to be made available.

This value is large enough for a standard HADB node that uses default values. If you change the default values, you should consider changing these values, as well.

To make these changes permanent, add those lines to `/etc/rc.local` on your Linux machine. (In modern Redhat versions of Linux, you can also modify `sysctl.conf` to set the kernel parameters.)

in modern Redhat versions

Tuning the Application Server for High-Availability

This section discusses how you can configure the high availability features of Application Server for your application.

Configuring and Tuning the Application Server

This section discusses the following topics:

- [Implications of Persistence Frequency on HTTP Session Performance](#)
- [Implications of Persistence Scope on HTTP Session Performance.](#)
- [Impact of checkpointing on Stateful Session Bean performance](#)
- [Configuring the JDBC Connection Pool](#)
- [Impact of Session Size on Performance](#)
- [Load Balancer Configuration](#)

To ensure highly available web applications with persistent session data, Application Server provides a backend store to save the `HttpSession` data. As such there is a overhead involved in saving and reading the data back from HADB. Understanding, the different schemes of session persistence and its impact on performance and availability will help you to make an appropriate decision in configuring Application Server for deployment.

It is recommended that a 1:2 ratio be maintained between the application server instances and HADB nodes. For every application server instance you need to have two HADB nodes.

Implications of Persistence Frequency on HTTP Session Performance

Application Server provides HTTP session persistence and thus failover by writing the session data to HADB. The frequency at which this is written to HADB is controlled and configured by specifying the persistence frequency to one of the following values:

- `web-method`
- `time-based`

Specify the persistence frequency in the `server.xml` in the following way. This entry is available under the following tree:

```
<session-config>
  <session-manager>
    <store-properties>
      <property name="persistenceFrequency" value="web-method"/>
```

The valid values for the `persistenceFrequency` are `web-method` and `time-based`.

Assuming that all the other parameters related to the server and the application remain same, `time-based` provides better performance than `web-method`. Availability is less in `time-based` when compared with `web-method`. This is because the session state is written to the persistent store (HADB) at the time interval that is configured via `reapIntervalSeconds1` (default is 60 sec). If the server instance fails within that interval, the updates made to the session state from the last time the session information is written to HADB are lost.

web-method as the Persistence Frequency

In cases where `web-method` is specified as the persistence frequency mechanism, even before the server response to the client request is sent, the server writes the current HTTP session state to HADB. Depending on the size of the data that needs to be persisted, the response time varies. This mode of persistence frequency is recommended for applications where availability is very critical and some performance degradation is acceptable. For more details on `web-method` as the persistence frequency, see the “Configuring Session Persistence” chapter in the *Application Server Administrator’s Guide*.

time-based as the persistence Frequency

In cases where `time-based` session is specified as the persistence frequency mechanism, the session persistence happens at the value specified in `reapIntervalSeconds`. By default, the value for `reapIntervalSeconds` is set to 60 seconds. There is a thread with configurable `reapInterval` at which time the thread wakes up and iterates over valid sessions in memory and saves the session data for the remaining valid sessions. This mode of persistence frequency is recommended when performance is critical and availability is not. The response to clients are not held back by performing the Save operation to the HADB. The default value can be changed under the `<manager-properties>` tag in the following way.

```
<property name="reapIntervalSeconds" value="valueInSeconds"/>
```

Summary

The supported schemes for persistence frequency are `web-method` and `time-based`. In terms of performance ranking, `time-based` performs better than `web-method`. If you are willing to trade availability during `reapIntervalSeconds`, it is recommended that you use `time-based` scheme as the persistence-frequency.

Implications of Persistence Scope on HTTP Session Performance

The Application Server allows the deployer to specify the scope of the persistence in addition to persistence frequency.

The valid persistence-scope values are:

- `session`
- `modified-session`
- `modified-attribute`

For more information, see [“Implications of Persistence Frequency on HTTP Session Performance” on page 138](#).

An example for specifying the persistence scope is shown in the following code:

```
<session-config>
    <session-manager persistence-type="ha">
        <manager-properties>
            <property name="persistenceFrequency"
                value="time-based"/>
        </manager-properties>
        <store-properties>
```

```

        <property name="persistenceScope" value="session"/>
    </store-properties>
</session-manager>
<session-properties>
</session-properties>
</session-config>

```

session as Persistence Scope

When `session` is specified as the persistence scope, the entire session data regardless of whether it is modified or not is written to the HADB. This ensures that the data in the backend store is always current. There is degradation of performance since all the session data needs to be persisted for every request.

modified-session as the persistence scope

When `modified-session` is specified as the persistence frequency mechanism, the server examines the state of the HTTP session. If and only if the data appears to be modified, the session persistence to HADB occurs. The `modified-session` is better than `session` because calls to HADB to persist data happens only when the session is modified.

modified-attribute as the persistence scope

The `modified-attribute` mode of persistence scope is useful as there are no cross references for the attributes and the application uses `setAttribute` and `getAttribute` semantics to manipulate HTTP session data.

Applications written this way can take advantage of this to obtain better performance.

For detailed description of different persistence scopes, see the “Configuring Session Persistence” chapter in the *Application Server Administrator’s Guide*.

Impact of checkpointing on Stateful Session Bean performance

Checkpointing ensures that a stateful bean state is saved into a highly available persistent store (HADB) so that in the event of failure of the server instance that is servicing requests using this SFSB can be failed over to another available instance in the cluster and the bean state can be recovered.

The size of the data that is being checkpointed and the frequency at which checkpointing happens determines the additional overhead in response time for a given client interaction.

From a performance point of view, checkpointing should be explicitly specified for only those methods that alter the bean state significantly, by adding the `<checkpointed-methods>` tag in the `sun-ejb-jar.xml` file. For more details, refer to the *Developer's Guide to Enterprise JavaBeans Technology*.

Configuring the JDBC Connection Pool

The Application Server relies on the JDBC connectivity to the backend persistent store (HADB) to store and retrieve the HTTP session data. Hence, it is imperative that the connection pool be configured in the best possible way to ensure that all Read-Write operations from the HADB occur faster.

The connection pool can be configured by running the `cladmin` script.

For more information on configuring the JDBC connection pool, see [“JDBC Connection Pool Tuning” on page 73](#). A more detailed discussion can be found in the *Application Server, Administrator's Guide*, chapter, “Monitoring the Application Server.”

The connection pool configuration settings are as follows:

```
<jdbc-connection-pool steady-pool-size="8" max-pool-size="16"
max-wait-time-in-millis="60000" pool-resize-quantity="2"
idle-timeout-in-seconds="300" is-isolation-level-guaranteed="true"
is-connection-validation-required="true"
connection-validation-method="meta-data" fail-all-connections="false"
datasource-classname="com.sun.hadb.jdbc.ds.HadbDataSource" name="CluJDBC"
transaction-isolation-level="repeatable-read">

  <description>HADB Connection pool configuration</description>

  <property value="test" name="password"/>
  <property value="test" name="username"/>

  <property value="host1:15105,host2:15125" name="serverList"/>
  <property name="maxStatements" value="30"/>
  <property name="cacheDatabaseMetaData" value="false"/>
  <property name="eliminateRedundantEndTransaction" value="true"/>
</jdbc-connection-pool>
```

For optimal performance, it is recommended that the pool contain eight to 16 connections per node. For example, if you have four nodes configured, then the steady-pool size must be set to 32 and the max-pool-size must be 64.

```
connection-validation-method="meta-data"
```

```
<property name="maxStatements" value="30"/>
<property name="cacheDatabaseMetaData" value="false"/>
<property name="eliminateRedundantEndTransaction" value="true"/>
```

The following settings must be adhered to.

is-connection-validation	True
connection-validation-method	meta-data
cacheDatabaseMetaData	False
eliminateRedundantEndTransaction	True
transaction-isolation-level	repeatable-read

By default, the connection pool is configured as follows. For more information on configuring the JDBC connection pool, see [“JDBC Connection Pool Tuning” on page 73](#). A more detailed discussion can be found in the *Application Server Administrator’s Guide*, chapter, “Monitoring the Application Server.”

NOTE The values of `idle-timeout-in-seconds` and also `pool-resize-quantity` must be adjusted based on the monitoring statistics.

Impact of Session Size on Performance

It is critical to be aware of the impact of HTTP session size on performance. Performance has an inverse relationship with the size of the session data that needs to be persisted. Session data is stored in HADB in a serialized manner. There is an overhead in serializing the data and inserting it as a BLOB and also deserializing it for retrieval.

In our tests we have observed that for upto 24KB session size, the performance remains unchanged. When the session size reaches 100KB and above, and if the same back end store is used for the same number of connections, then throughput drops by 90%.

Pay sufficient attention while determining the HTTP session size. If you are creating large HTTP session objects, calculate the HADB nodes as discussed in [“Tuning HADB” on page 125](#).

Load Balancer Configuration

Application Server provides a load balancer plugin for the Sun ONE Web Server that can balance the load of requests among multiple instances which are part of the cluster. For more information on configuring the load balancer see the *Application Server Administrator's Guide*, "Configuring Load Balancer."

Before you tune the parameters in the load balancer configuration file (`loadbalancer.xml`), see the *Web Server Performance and Tuning Guide* for information on how to tune the web server for best results.

NOTE It is assumed in the following section that the web server is tuned effectively to service the incoming requests.

The load balancer periodically checks all the configured Application Server instances that are marked as unhealthy, based on the values specified in the `health-checker` element. Enabling the `health-checker` is optional. If the `health-checker` is not enabled, periodic health check of unhealthy instances is not performed.

The load balancer's health check mechanism communicates with the application server instance using HTTP. The health checker sends an HTTP request to the URL specified and waits for a response. The status code in the HTTP response header should be between 100 and 500 to consider the instance to be healthy.

To enable the health checker, edit the following properties in the `loadbalancer.xml` file:

url

Specifies the listener's URL that the load balancer checks to determine its state of health.

interval-in-seconds

Specifies the interval at which health checks of instances occur. The default is 30 seconds.

timeout-in-seconds

Specifies the timeout interval within which a response must be obtained for a listener to be considered healthy. The default is 10 seconds.

If the typical response from the server takes n number of seconds and under peak load takes m number of seconds, then it is suggested to configure the health checker as shown below:

```
<health-checker url="http://hostname.domain:port" interval-in-seconds="n"  
timeout-in-seconds="m+n"/>
```

For more information, see the “Configuring Load Balancer” chapter in the *Application Server Administrator’s Guide*.

Common Performance Problems

This section discusses a few common web site performance problems to check for:

- [check-acl Server Application Functions](#)
- [Low-Memory Situations](#)
- [Under-Throttled Server](#)
- [Cache Not Utilized](#)
- [Keep-Alive Connections Flushed](#)
- [Log File Modes](#)
- [Additional Resources](#)

NOTE You will find the *Troubleshooting Guide* useful in resolving common problems with the Application Server.

check-acl Server Application Functions

For optimal performance of your server, use ACLs only when required.

The default server is configured with an ACL file containing the default ACL allowing write access to the server only to `all`, and an es-internal ACL for restricting write access for `anybody`. The latter protects the manuals, icons, and search UI files in the server.

The default `obj.conf` file has `NameTrans` lines mapping the directories that need to be read-only to the es-internal object, which in turn has a check-acl SAF for the es-internal ACL.

The default object also contains a check-acl SAF for the "default" ACL.

You can improve your server's performance by removing the `acis` properties from virtual server tags in `<instance_name>-obj.conf`. This stops any ACL processing.

You can also improve performance by removing the `check-acl` SAF from the default object for URIs that are not protected by ACLs.

Low-Memory Situations

If you need Application Server to run in low-memory situations, reduce the thread limit to a bare minimum by lowering the value of `RqThrottle`. Also, you may want to reduce the maximum number of processes that the Application Server will spawn by lowering the value of the `MaxProcs` value.

Under-Throttled Server

The server does not allow the number of active threads to exceed the thread limit value. If the number of simultaneous requests reaches that limit, the server stops servicing new connections until the old connections are freed up. This can lead to increased response time.

In Application Server, the server's default `RqThrottle` value is 128. If you want your server to process more requests concurrently, you need to increase the `RqThrottle` value.

The symptom of an under-throttled server is a server with a long response time. Making a request from a browser establishes a connection fairly quickly to the server, but on under-throttled servers it may take a long time before the response comes back to the client.

The best way to tell if your server is being throttled is to see if the number of active sessions is close to, or equal to, the maximum number allowed via `RqThrottle`. To do this, see [Maximum Simultaneous Requests](#).

Cache Not Utilized

If the cache is not utilized, your server is not performing optimally. Since most sites have lots of GIF or JPEG files that should always be cacheable, you need to use your cache effectively.

Some sites, however, do almost everything through CGIs, SHTML, or other dynamic sources. Dynamic content is generally not cacheable, and inherently yields a low cache hit rate. Don't be too alarmed if your site has a low cache hit rate. The most important thing is that your response time is low. You can have a very low cache hit rate and still have very good response time. As long as your response time is good, you may not care that the cache hit rate is low.

Check your Hit Ratio using statistics from `perfdump` or the Monitor Current Activity page of the web-based Admin Console. The hit ratio is the percentage of times the cache was used with all hits to your server. A good cache hit rate is anything above 50%. Some sites may even achieve 98% or higher.

In addition, if you are doing a lot of CGI or NSAPI calls, you may have a low cache hit rate. If you have custom NSAPI functions, you may have a low cache hit rate.

Keep-Alive Connections Flushed

A web site that might be able to service 75 requests per second without keep-alive connections, may be able to do 200-300 requests per second when keep-alive is enabled. Therefore, as a client requests various items from a single page, it is important that keep-alive connections are being used effectively. If the `KeepAliveCount` exceeds the `MaxKeepAliveConnections`, subsequent keep-alive connections will be closed, or 'flushed', instead of being honored and kept alive.

Check the `KeepAliveFlushes` and `KeepAliveHits` values using statistics from `perfdump` or the Monitor Current Activity page of the web-based Admin Console. On a site where keep-alive connections are running well, the ratio of `KeepAliveFlushes` to `KeepAliveHits` is very low. If the ratio is high (greater than 1:1), your site is probably not utilizing keep-alive connections as well as it could.

To reduce keep-alive flushes, increase the `MaxKeepAliveConnections` value in the `init.conf` file or the web-based Admin Console. The default value is 200. By raising the value, you keep more waiting keep-alive connections open.

CAUTION On Unix systems, if you increase the `MaxKeepAliveConnections` value too high, the server can run out of open file descriptors. Typically 1024 is the limit for open files on Unix, so increasing this value above 500 is not recommended.

Log File Modes

Keeping the log files on verbose mode can have a significant affect of performance. You can set `LogVerbose` to any level higher than `FINE` using the web-based Admin Console.

Additional Resources

Places to go for additional information:

- “Monitoring Current Activity Using the perfdump Utility” on page 37
- SE Toolkit
<http://www.setoolkit.com>
Gets system performance data in a color-coded GUI.
- GC Portal:
<http://java.sun.com/developer/technicalArticles/Programming/GCPortal/>
How to configure the vm parameters.
- Solaris Tunable Parameters reference manual.
<http://docs.sun.com/db/doc/806-7009>
- JDK performance
<http://java.sun.com/performance>
- Linux performance and other information:
 - <http://www.tldp.org>
 - <http://linuxperf.nl.linux.org/>
- Solaris information:
 - <http://www.sun.com/bigadmin/patches>
For performance patches.
 - <http://www.sun.com/bigadmin/>
For a variety of other resources.

Index

A

Acceptor Threads [42](#), [63](#)
AcceptTimeOut [66](#)
ACL [145](#)
ACL User Cache [60](#)
ACLCacheLifetime [61](#)
ACLGroupCacheSize [61](#)
aclis properties [146](#)
ACLUserCacheSize [61](#)
Address [42](#)
AddrLookups [52](#)
Alternate Thread Library [65](#)
Application Design and Implementation [17](#)
Arrays [21](#)
Average Queuing Delay [41](#)

B

Busy Functions [52](#)

C

Cache [48](#)
cache hit rate [147](#)
Cache Not Utilized [146](#)
cached bean [28](#)

CacheEntries [48](#), [51](#)
cache-idle-timeout-in-seconds [81](#)
cache-resize-quantity [81](#)
CGIStub [66](#)
check-acl Server Application [145](#)
Commit options [88](#)
commit-option [82](#)
Connection Pool Tuning [71](#)
ConnQueueSize [41](#)
context factory [96](#)
CORBA [97](#)
Current /peak /limit [40](#)

D

data volume [125](#)
datasource-classname [72](#)
deserialization [21](#)
DNS Cache [50](#)
DNS Lookups [52](#)

E

EJB [26](#)
EJB Cache [83](#)
EJB Container [77](#), [79](#)
EJB Descriptor Properties [80](#)

EJB Pool and Cache [28](#)
Entity beans [84](#)
entity beans [28](#)
es-internal object [145](#)
Examining IIOP Messages [99](#)

F

fail-all-connections [73](#)
File Cache [56](#)
finalizers [23](#)
find-pathinfo-forward [68](#)
Footprint [108](#)
Fragmented Messages [100](#)

G

Garbage Collector [106](#)
generational object memory [106](#)

H

HADB [125](#)
HADB data device size [125](#)
high-availability database [125](#)
Hit Ratio [49](#), [147](#)
HitRatio [51](#)
HotSpot [107](#), [112](#)
HTTP Listener Information
 tuning [42](#)
Http Listener Information [41](#)
HTTP server instances [33](#)
HttpSession [75](#)

I

Idle/Peak/Limit [49](#)
idle-thread-timeout-in-seconds [96](#)
idle-timeout-in-seconds [72](#)
Inter-ORB Communication Infrastructure [94](#)
iop-listener [96](#)
is-cache-overflow-allowed [81](#)
is-connection-validation-required [73](#)
is-isolation-level-guaranteed [73](#)

J

J2EE Programming Guidelines [24](#)
Java Coding Guidelines [21](#)
Java Heap [110](#)
Java Performance [65](#)
JDBC [30](#)
JDBC Connection Pools [71](#)
JMS [31](#)
JSP [24](#)
JSP and Servlet Tuning [75](#)

K

Keep-Alive Connections [147](#)
Keep-Alive/Persistent Connection [43](#)
KeepAliveCount [45](#), [147](#)
KeepAliveFlushes [45](#), [147](#)
KeepAliveHits [45](#), [147](#)
KeepAliveQueryMeanTime [46](#)
KeepAliveThreads [44](#)
KeepAliveTimeout [46](#)

L

light weight processes [105](#)

Local Interfaces [100](#)
 Log File Modes [148](#)
 LogVerbose [148](#)
 long response time [146](#)
 LookupsInProgress [52](#)
 low-memory situations [146](#)
 lwps [105](#)

M

max-cache-size [81](#)
 max-connections [96](#)
 Maximum Age [49](#)
 MaxKeepAliveConnections [147](#)
 MaxNewSize [111](#)
 max-pool-size [72](#), [81](#)
 MaxProcs [146](#)
 max-thread-pool-size [96](#)
 max-wait-time-in-millis [72](#), [81](#)
 Message driven beans [86](#)
 message-fragment-size [95](#)

N

name [72](#)
 NameLookups [52](#)
 NameTrans [145](#)
 NewRatio [111](#)
 NewSize [111](#)
 -nolocalstubs flag [27](#)
 nostat [69](#)
 NSAPI functions [147](#)
 NSAPI performance profiling [36](#)

O

open file descriptors [147](#)

operating system
 tuning [115](#)
 ORB Properties [95](#)
 ORB tuning [92](#)

P

Pass-by-reference [86](#)
 Pass-by-value [86](#)
 Pauses [108](#)
 perfdump [34](#), [37](#), [40](#), [42](#), [47](#)
 Performance Buckets [53](#)
 Performance Report [54](#)
 pooled bean [28](#)
 pool-idle-timeout-in-seconds [81](#)
 pool-resize-quantity [72](#), [80](#)
 Pre-compiled JSPs [65](#)
 profiling [36](#)

R

Read only beans [85](#)
 refresh-period-in-seconds [82](#)
 Remote vs Local interfaces [86](#)
 removal-timeout-in-seconds [81](#)
 res-type [72](#)
 rlim_fd_cur [115](#)
 rlim_fd_max [115](#)
 rmic [27](#)
 rpm [14](#)
 RqThrottle [146](#)

S

Safety Margins [17](#)
 separate disks [125](#)
 multiple data devices [125](#)

- serialization [21](#)
- Server ORB Thread Pool [95](#)
- server.xml [72](#)
- servlets [24](#)
- Session Creation [47](#)
- showrev [14](#)
- simultaneous requests [146](#)
- Solaris JDK [107](#)
- sq_max_size [116](#)
- Stateful session beans [84](#)
- statefull session beans [28](#)
- Stateless session beans [85](#)
- stateless session beans [28](#)
- stats-xml [34](#)
- steady-pool-size [72, 80](#)
- steady-thread-pool-size [95](#)
- storing persistent session state [125](#)
- Sun customer support [13](#)
- sun.rmi.dgc.client.gcInterval [110](#)
- Survivor Ratio Sizing [112](#)
- System.gc() [109](#)

T

- tcp_close_wait_interval [116](#)
- tcp_conn_hash_size [117](#)
- tcp_conn_req_max_q [116](#)
- tcp_conn_req_max_q0 [116](#)
- tcp_ip_abort_interval [116](#)
- tcp_keepalive_interval [116](#)
- tcp_recv_hiwat [116](#)
- tcp_rexmit_interval_initial [116](#)
- tcp_rexmit_interval_max [116](#)
- tcp_rexmit_interval_min [116](#)
- tcp_slow_start_initial [116](#)
- tcp_smallest_anon_port [116](#)
- tcp_time_wait_interval [116](#)
- tcp_xmit_hiwat [116](#)
- thread limit [146](#)
- Thread Pool Sizing [98](#)

- Thread Pools [49](#)
- Throughput [108](#)
- Total Connections Queued [41](#)
- transaction frequency [125](#)
- Transaction management for CMT [87](#)
- transaction-isolation-level [72](#)
- Transactions [29](#)
- tuning Solaris TCP/IP settings [115](#)

U

- ulimit [117](#)
- Under-Throttled Server [146](#)
- update-interval [36](#)
- UseNativePoll [46](#)
- User Load [16](#)

V

- victim-selection-policy [82](#)
- virtual-servers [36](#)

W

- Work Queue Length [50](#)

X

- Xms [111](#)
- Xmx [111](#)
- XX
 - +DisableExplicitGC [109](#)
 - MaxHeapFreeRatio [111](#)
 - MaxPermSize [109](#)
 - MinHeapFreeRatio [111](#)