

Sun Java System Web Server 7.0 Update 8 Performance Tuning, Sizing, and Scaling Guide



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

Part No: 821-1489
February 2010

Copyright 2010 Sun Microsystems, Inc. 4150 Network Circle, Santa Clara, CA 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 U.S. patents or pending patent applications in the U.S. and in other countries.

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.

This distribution may include materials developed by third parties.

Parts of the product may be derived from Berkeley BSD systems, licensed from the University of California. UNIX is a registered trademark in the U.S. and other countries, exclusively licensed through X/Open Company, Ltd.

Sun, Sun Microsystems, the Sun logo, the Solaris logo, the Java Coffee Cup logo, docs.sun.com, JavaServer Pages, JSP, JVM, JDBC, Java HotSpot, Java, and Solaris are trademarks or registered trademarks of Sun Microsystems, Inc. or its subsidiaries 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 an architecture developed by Sun Microsystems, Inc. Netscape is a trademark or registered trademark of Netscape Communications Corporation in the United States and other countries.

The OPEN LOOK and Sun™ Graphical User Interface was developed by Sun Microsystems, Inc. for its users and licensees. Sun acknowledges the pioneering efforts of Xerox in researching and developing the concept of visual or graphical user interfaces for the computer industry. Sun holds a non-exclusive license from Xerox to the Xerox Graphical User Interface, which license also covers Sun's licensees who implement OPEN LOOK GUIs and otherwise comply with Sun's written license agreements.

Products covered by and information contained in this publication are controlled by U.S. Export Control laws and may be subject to the export or import laws in other countries. Nuclear, missile, chemical or 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 2010 Sun Microsystems, Inc. 4150 Network Circle, Santa Clara, CA 95054 U.S.A. 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 plusieurs brevets américains ou des applications de brevet en attente aux États-Unis et dans d'autres pays.

Cette distribution peut comprendre des composants développés par des tierces personnes.

Certaines composants de ce produit peuvent être dérivées du logiciel Berkeley BSD, licenciés par l'Université de Californie. UNIX est une marque déposée aux États-Unis et dans d'autres pays; elle est licenciée exclusivement par X/Open Company, Ltd.

Sun, Sun Microsystems, le logo Sun, le logo Solaris, le logo Java Coffee Cup, docs.sun.com, JavaServer Pages, JSP, JVM, JDBC, Java HotSpot, Java et Solaris sont des marques de fabrique ou des marques déposées de Sun Microsystems, Inc., ou ses filiales, aux États-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 États-Unis et dans d'autres pays. Les produits portant les marques SPARC sont basés sur une architecture développée par Sun Microsystems, Inc. Netscape est une marque de Netscape Communications Corporation aux États-Unis et dans d'autres pays.

L'interface d'utilisation graphique OPEN LOOK et Sun a été développée par Sun Microsystems, Inc. pour ses utilisateurs et licenciés. Sun reconnaît les efforts de pionniers de Xerox pour la recherche et le développement du concept des interfaces d'utilisation visuelle ou graphique pour l'industrie de l'informatique. Sun détient une licence non exclusive de Xerox sur l'interface d'utilisation graphique Xerox, cette licence couvrant également les licenciés de Sun qui mettent en place l'interface d'utilisation graphique OPEN LOOK et qui, en outre, se conforment aux licences écrites de Sun.

Les produits qui font l'objet de cette publication et les informations qu'il contient sont régis par la législation américaine en matière de contrôle des exportations et peuvent être soumis au droit d'autres pays dans le domaine des exportations et importations. Les utilisations finales, ou utilisateurs finaux, pour des armes nucléaires, des missiles, des armes chimiques ou biologiques ou pour le nucléaire maritime, directement ou indirectement, sont strictement interdites. Les exportations ou réexportations vers des pays sous embargo des États-Unis, ou vers des entités figurant sur les listes d'exclusion d'exportation américaines, y compris, mais de manière non exclusive, 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 CONTREFACON.

Contents

Preface	11
1 Performance and Monitoring Overview	17
Performance Issues	17
Configuration	18
Virtual Servers	18
Server Farms	19
64-Bit Servers	19
SSL Performance	19
Monitoring Server Performance	20
About Statistics	21
Monitoring Current Activity Using the Admin Console	23
Monitoring Current Activity Using the CLI	24
Monitoring Current Activity Using stats.xml	27
Monitoring Current Activity Using perfdump	29
Monitoring Current Activity Using the Java ES Monitoring Console	35
2 Tuning Sun Java System Web Server	37
General Tuning Tips	37
Understanding Threads, Processes, and Connections	38
Connection-Handling Overview	38
Custom Thread Pools	40
The Native Thread Pool	41
Process Modes	42
Mapping Web Server 6.1 Tuning Parameters to Web Server 7.0	44
Using Monitoring Data to Tune Your Server	46
Connection Queue Information	48

HTTP Listener (Listen Socket) Information	50
Keep-Alive Information	51
Session Creation and Thread Information	56
File Cache Statistics Information	58
Thread Pool Information	65
DNS Cache Information	68
Java Virtual Machine (JVM) Information	70
Web Application Information	71
JDBC Resource Information	72
Tuning the ACL User Cache	77
Tuning Java Web Application Performance	78
Using Precompiled JSPs	78
Using Servlet/JSP Caching	79
Configuring the Java Security Manager	79
Configuring Class Reloading	79
Avoiding Directories in the Classpath	79
Configuring the Web Application's Session Settings	80
Tuning CGI Stub Processes (UNIX/Linux)	81
Using find-pathinfo-forward	81
Using nostat	82
Using Busy Functions	83
Using Large Pages Support	83
Tuning Your Web Application	84
Java Programming Guidelines	85
Avoid Serialization and Deserialization	85
Use StringBuffer to Concatenate Strings	85
Assign null to Variables That Are No Longer Needed	85
Declare Methods as final Only If Necessary	86
Declare Constants as static final	86
Avoid Finalizers	86
Declare Method Arguments final	86
Synchronize Only When Necessary	86
Use DataHandlers for SOAP Attachments	87
Java Server Page and Servlet Tuning	87
Suggested Coding Practices	87
Tuning Web Container Within Web Server 7.0	88

Deployment Settings	89
J2SE 5.0 Monitoring Tools	93
Binary Logging Header	93
3 Common Performance Problems	95
check-acl Server Application Functions	95
Low-Memory Situations	96
Too Few Threads	96
Cache Not Utilized	97
Keep-Alive Connections Flushed	97
Large Memory Footprint	98
Log File Modes	98
Tuning of File Descriptors	98
4 Platform-Specific Issues and Tips	101
Solaris Platform-Specific Issues	101
Files Open in a Single Process (File Descriptor Limits)	101
Failure to Connect to HTTP Server	102
Connection Refused Errors	103
Tuning TCP Buffering	103
Using the Solaris Network Cache and Accelerator (SNCA)	103
Solaris File System Tuning	105
High File System Page-In Rate	105
Reduce File System Housekeeping	105
Long Service Times on Busy Disks or Volumes	105
Solaris Platform-Specific Performance Monitoring	106
Short-Term System Monitoring	106
Long-Term System Monitoring	107
“Intelligent” Monitoring	107
Solaris 10 Platform-Specific Tuning Information	107
Tuning Solaris for Performance Benchmarking	108
Tuning UltraSPARC T1–Based Systems for Performance Benchmarking	109
Tuning Operating System and TCP Settings	109
Disk Configuration	110
Network Configuration	110

Web Server Start Options	111
5 Sizing and Scaling Your Server	113
64-Bit Server	113
Processors	113
Memory	114
Drive Space	114
Networking	114
6 Scalability Studies	115
Study Goals	115
Study Conclusion	116
Hardware	116
Software	117
Configuration and Tuning	117
Network Configuration	118
Web Server Tuning	119
Performance Tests and Results	120
Static Content Test	120
Dynamic Content Test: Servlet	122
Dynamic Content Test: C CGI	123
Dynamic Content Test: Perl CGI	125
Dynamic Content Test: NSAPI	126
PHP Scalability Tests	127
SSL Performance Test: Static Content	130
SSL Performance Test: Perl CGI	132
SSL Performance Test: C CGI	133
SSL Performance Test: NSAPI	134
E-Commerce Web Application Test	135
Index	139

Figures

FIGURE 2-1 Web Server Connection Handling 39

Tables

TABLE 1-1	Methods of Monitoring Performance	20
TABLE 2-1	Parameter Mapping to server.xml	45
TABLE 2-2	Connection Queue Statistics	48
TABLE 2-3	Keep-Alive Statistics	52
TABLE 2-4	File Cache Statistics	59
TABLE 2-5	Thread Pools Statistics	66
TABLE 2-6	DNS Cache Statistics	68
TABLE 2-7	Java Virtual Machine (JVM) Statistics	70
TABLE 2-8	Web Application Statistics	72
TABLE 2-9	JDBC Resource Statistics	73
TABLE 4-1	Tuning Solaris for Performance Benchmarking	108
TABLE 4-2	Tuning 64-bit Systems for Performance Benchmarking	109
TABLE 6-1	Web Server Tuning Settings	119
TABLE 6-2	SSL Session Cache Tuning Settings	119
TABLE 6-3	File Cache Configuration	121
TABLE 6-4	Static Content Scalability	121
TABLE 6-5	JVM Tuning Settings	122
TABLE 6-6	Dynamic Content Test: Servlet Scalability	123
TABLE 6-7	CGI Tuning Settings	124
TABLE 6-8	Dynamic Content Test: C CGI Scalability	124
TABLE 6-9	CGI Tuning Settings	125
TABLE 6-10	Dynamic Content Test: Perl CGI Scalability	126
TABLE 6-11	Dynamic Content Test: NSAPI Scalability	127
TABLE 6-12	Tuning Settings for FastCGI Plug-in Test	128
TABLE 6-13	PHP Scalability with Fast CGI	128
TABLE 6-14	NSAPI Plug-in Configuration for PHP	129
TABLE 6-15	PHP Scalability with NSAPI	130
TABLE 6-16	SSL Performance Test: Static Content Scalability	131

TABLE 6-17	SSL Performance Test: Perl CGI Scalability	132
TABLE 6-18	SSL Performance Test: C CGI Scalability	133
TABLE 6-19	SSL Performance Test: NSAPI Scalability	134
TABLE 6-20	Performance Test Pass Criteria	137
TABLE 6-21	E-Commerce Web Application Scalability	137

Preface

This guide discusses adjustments you can make that may improve the performance of Sun Java System Web Server (henceforth known as Web Server). The guide provides tuning, scaling, and sizing tips and suggestions; possible solutions to common performance problems; and data from scalability studies. It also addresses miscellaneous configuration and platform-specific issues.

Who Should Use This Book

This guide is intended for advanced administrators only. Be sure to read this guide and other relevant server documentation before making any changes. Be very careful when tuning your server, and always back up your configuration files before making any changes.

Web Server Documentation Set

The Web Server documentation set describes how to install and administer the Web Server. You can access the Web Server 7.0 Update 8 documentation at <http://docs.sun.com/coll/1653.8>.

The Sun Java System Web Server documents are now in wiki format at <http://wikis.sun.com/display/WebServerdocs/Home>. This wiki is intended to promote collaboration and contribution on documentation content for Web Server. You are welcome to contribute, by posting your comments or by directly editing the wiki page, as long as the content is relevant to an appropriate standard.

For an introduction to Web Server , refer to the books in the order in which they are listed in the following table.

TABLE P-1 Books in the Web Server Documentation Set

Documentation Title	Contents
Sun Java System Web Server Documentation Wiki	This wiki is intended to promote collaboration and contribution on documentation content for Web Server.
Sun Java System Web Server 7.0 Update 8 Documentation Center	Web Server documentation topics organized by tasks and subject

TABLE P-1 Books in the Web Server Documentation Set (Continued)

Documentation Title	Contents
<i>Sun Java System Web Server 7.0 Update 8 Release Notes</i>	<ul style="list-style-type: none"> ■ Late-breaking information about the software and documentation ■ Supported platforms and patch requirements for installing Web Server
<i>Sun Java System Web Server 7.0 Update 8 Installation and Migration Guide</i>	<p>Performing installation and migration tasks:</p> <ul style="list-style-type: none"> ■ Installing Web Server and its various components, ■ Migrating data from Sun ONE Web Server 6.0 or 6.1 to Sun Java System Web Server 7.0
<i>Sun Java System Web Server 7.0 Update 8 Administrator's Guide</i>	<p>Performing the following administration tasks:</p> <ul style="list-style-type: none"> ■ Using the Administration GUI and command-line interface ■ Configuring server preferences ■ Using server instances ■ Monitoring and logging server activity ■ Using certificates and public key cryptography to secure the server ■ Configuring access control to secure the server ■ Using Java™ Platform Enterprise Edition (Java EE) security features ■ Deploying applications ■ Managing virtual servers ■ Defining server workload and sizing the system to meet performance needs ■ Searching the contents and attributes of server documents, and creating a text search interface ■ Configuring the server for content compression ■ Configuring the server for web publishing and content authoring using WebDAV
<i>Sun Java System Web Server 7.0 Update 8 Developer's Guide</i>	<p>Using programming technologies and APIs to do the following:</p> <ul style="list-style-type: none"> ■ Extend and modify Sun Java System Web Server ■ Dynamically generate content in response to client requests and modify the content of the server
<i>Sun Java System Web Server 7.0 Update 8 NSAPI Developer's Guide</i>	Creating custom Netscape Server Application Programmer's Interface (NSAPI) plug-ins
<i>Sun Java System Web Server 7.0 Update 8 Developer's Guide to Java Web Applications</i>	Implementing Java Servlets and JavaServer Pages™ (JSP™) technology in Sun Java System Web Server

TABLE P-1 Books in the Web Server Documentation Set (Continued)

Documentation Title	Contents
<i>Sun Java System Web Server 7.0 Update 8 Administrator's Configuration File Reference</i>	Editing configuration files
<i>Sun Java System Web Server 7.0 Update 8 Performance Tuning, Sizing, and Scaling Guide</i>	Tuning Sun Java System Web Server to optimize performance
<i>Sun Java System Web Server 7.0 Update 8 Troubleshooting Guide</i>	Troubleshooting Web Server
<i>Sun Java System Web Server 7.0 Update 8 CLI Reference Manual</i>	Administration commands that allow you to administer the Web Server through the CLI

Related Books

The URL for all documentation about Sun Java Enterprise System (Java ES) and its components is <http://docs.sun.com/coll/1286.3>.

Default Paths and File Names

The following table describes the default paths and file names that are used in this book.

TABLE P-2 Default Paths and File Names

Placeholder	Description	Default Value
<i>install-dir</i>	Represents the base installation directory for Web Server	<p>Sun Java Enterprise System (Java ES) installations on the Solaris™ platform:</p> <p><i>/opt/SUNWwbsvr7</i></p> <p>Java ES installations on the Linux and HP-UX platform:</p> <p><i>/opt/sun/webserver/</i></p> <p>Java ES installations on the Windows platform:</p> <p><i>system-drive:\Program Files\Sun\JavaES5\WebServer7</i></p> <p>Other Solaris, Linux, and HP-UX installations, non-root user:</p> <p><i>home-directory/sun/webserver7</i></p> <p>Other Solaris, Linux, and HP-UX installations, root user:</p> <p><i>/sun/webserver7</i></p> <p>Windows, all installations:</p> <p><i>system-drive:\Program Files\Sun\WebServer7</i></p>

TABLE P-2 Default Paths and File Names (Continued)

Placeholder	Description	Default Value
<i>instance-dir</i>	Directory that contains the instance-specific subdirectories.	<p>For Java ES installations, the default location for instances on Solaris:</p> <p><i>/var/opt/SUNWwbsvr7</i></p> <p>For Java ES installations, the default location for instances on Linux and HP-UX:</p> <p><i>/var/opt/sun/webserver7</i></p> <p>For Java ES installations, the default location for instance on Windows:</p> <p><i>system-drive:\Program Files\Sun\JavaES5\WebServer7</i></p> <p>For stand-alone installations, the default location for instance on Solaris, Linux, and HP-UX:<i>install-dir</i></p> <p>For stand-alone installations, the default location for instance on Windows:</p> <p><i>system-drive:\Program Files\sun\WebServer7</i></p>

Typographic Conventions

The following table describes the typographic changes that are used in this book.

TABLE P-3 Typographic Conventions

Typeface	Meaning	Example
AaBbCc123	The names of commands, files, and directories, and onscreen computer output	<p>Edit your <i>.login</i> file.</p> <p>Use <i>ls -a</i> to list all files.</p> <p><i>machine_name%</i> you have mail.</p>
AaBbCc123	What you type, contrasted with onscreen computer output	<p><i>machine_name%</i> su</p> <p>Password:</p>
<i>AaBbCc123</i>	A placeholder to be replaced with a real name or value	<p>The command to remove a file is <i>rm filename</i>.</p>
<i>AaBbCc123</i>	Book titles, new terms, and terms to be emphasized (note that some emphasized items appear bold online)	<p>Read Chapter 6 in the <i>User's Guide</i>.</p> <p>A <i>cache</i> is a copy that is stored locally.</p> <p>Do <i>not</i> save the file.</p>

Symbol Conventions

The following table explains symbols that might be used in this book.

TABLE P-4 Symbol Conventions

Symbol	Description	Example	Meaning
[]	Contains optional arguments and command options.	ls [-l]	The -l option is not required.
{ }	Contains a set of choices for a required command option.	-d {y n}	The -d option requires that you use either the y argument or the n argument.
\${ }	Indicates a variable reference.	\${com.sun.javaRoot}	References the value of the com.sun.javaRoot variable.
-	Joins simultaneous multiple keystrokes.	Control-A	Press the Control key while you press the A key.
+	Joins consecutive multiple keystrokes.	Ctrl+A+N	Press the Control key, release it, and then press the subsequent keys.
→	Indicates menu item selection in a graphical user interface.	File → New → Templates	From the File menu, choose New. From the New submenu, choose Templates.

Documentation, Support, and Training

The Sun web site provides information about the following additional resources:

- [Documentation \(http://www.sun.com/documentation/\)](http://www.sun.com/documentation/)
- [Support \(http://www.sun.com/support/\)](http://www.sun.com/support/)
- [Training \(http://www.sun.com/training/\)](http://www.sun.com/training/)

Searching Sun Product Documentation

Besides searching Sun product documentation from the docs.sun.com web site, you can use a search engine by typing the following syntax in the search field:

```
search-term site:docs.sun.com
```

For example, to search for “Web Server,” type the following:

```
Web Server site:docs.sun.com
```

To include other Sun web sites in your search (for example, java.sun.com, www.sun.com, and developers.sun.com), use “sun.com” in place of “docs.sun.com” in the search field.

Third-Party Web Site References

Third-party URLs are referenced in this document and provide additional, related information.

Note – Sun is not responsible for the availability of third-party web sites mentioned in this document. Sun does not endorse and is not responsible or liable for any content, advertising, products, or other materials that are available on or through such sites or resources. Sun will not be responsible or liable for any actual or alleged damage or loss caused or alleged to be caused by or in connection with use of or reliance on any such content, goods, or services that are available on or through such sites or resources.

Sun Welcomes Your Comments

Sun is interested in improving its documentation and welcomes your comments and suggestions. To share your comments:

- Go to <http://docs.sun.com> and click Feedback.
- Go to <http://wikis.sun.com/display/WebServerdocs/Home> and post your comments or directly edit the wiki page.

Performance and Monitoring Overview

Sun Java System Web Server (hereafter Web Server) is designed to meet the needs of the most demanding, high-traffic sites in the world. It can serve both static and dynamically generated content. Web Server can also run in SSL mode, enabling the secure transfer of information.

This guide helps you to define your server workload and size a system to meet your performance needs. Because your environment is unique, the impacts of the suggestions provided in this guide will depend on your specific environment. Ultimately you must rely on your own judgement and observations to select the adjustments that are best for you.

This chapter provides a general discussion of server performance considerations, and more specific information about monitoring server performance.

This chapter includes the following topics:

- “Performance Issues” on page 17
- “Configuration” on page 18
- “Virtual Servers” on page 18
- “Server Farms” on page 19
- “64-Bit Servers” on page 19
- “SSL Performance” on page 19
- “Monitoring Server Performance” on page 20

Performance Issues

The first step toward sizing your server is to determine your requirements. Performance means different things to users than it means to webmasters. Users want fast response times (typically less than 100 milliseconds), high availability (no “connection refused” messages), and as much interface control as possible. Webmasters and system administrators, on the other hand, want to see high connection rates, high data throughput, and uptime approaching 100%. In addition, for virtual servers the goal might be to provide a targeted level of performance at different price points. You need to define what performance means for your particular situation.

Here are some areas to consider:

- The number of peak concurrent users
- Security requirements

Encrypting your Web Server's data streams with SSL makes an enormous difference to your site's credibility for electronic commerce and other security conscious applications, but it can also seriously impact your CPU load. For more information, see [“SSL Performance” on page 19](#).

- The size of the document tree
- Dynamic or static content

The content you serve affects your server's performance. A Web Server delivering mostly static HTML can run much faster than a server that must execute CGIs for every query.

Configuration

Certain tuning parameters are set at the configuration level, so that every server instance that is based on the configuration has the same tuning information. In addition, some monitoring information is available at the configuration level, so you can monitor the performance of all instances based on the configuration. However, the bulk of the monitoring information is available at the individual server instance, or virtual server level. If you are using a single Web Server instance per configuration, meaning your server is not part of a server farm, the configuration-level statistics show the information for the single server instance based on that configuration.

Virtual Servers

Virtual servers add another layer to the performance improvement process. Certain settings are tunable for the configuration, while others are based on an individual virtual server.

You can also use the quality of service (QoS) features to set resource utilization constraints for an individual virtual server. For example, you can use QoS features to limit the amount of bandwidth and the number of connections allowed for a virtual server. You can set these performance limits, track them, and optionally enforce them.

For more information about using the quality of service features, see [Sun Java System Web Server 7.0 Update 8 Administrator's Guide](#).

Server Farms

The clustering features of Web Server allow you to easily deploy to a server farm. Because all servers in a server farm share identical configurations, tuning is not done on a server-by-server basis.

64-Bit Servers

The performance for the 64-bit Web Server is not necessarily better than the performance for the 32-bit Web Server, but the 64-bit server scales better. Because the 32-bit Web Server process is confined to 4 GB of address space, it can run out of address space when attempting to support simultaneous sessions beyond a certain limit. Even if the host machine has available memory and CPU resources, the 32-bit Web Server might not be able to take advantage of it because of the address space limit. The 64-bit Web Server can run more applications and servlets than the 32-bit server. Also, the 64-bit Web Server can cache several GBs of static content, while the 32-bit Web Server is confined to 4 GB of address space.

In general, the tuning for the 64-bit Web Server is similar to the tuning for the 32-bit Web Server. The differences are mostly tuned at the operating system level. Tuning specifics are discussed in [“Tuning UltraSPARC T1-Based Systems for Performance Benchmarking” on page 109](#).

SSL Performance

SSL always has a significant impact on throughput, so for best performance minimize your use of SSL, or consider using a multi-CPU server to handle it.

For SSL, the Web Server uses the NSS library. However, there are other options available for SSL:

- If you are using the Solaris 10 operating system, kernel SSL (KSSL) is available. It does not contain all the algorithms available, as does NSS, but it often provides better performance.
- A cryptographic card hardware accelerator for SSL can also improve performance.
- If you are using the 64-bit Web Server on Solaris, you can use the cryptographic accelerator of the UltraSPARC T1 processor.

Monitoring Server Performance

Making the adjustments described in this guide without measuring their effects doesn't make sense. If you don't measure the system's behavior before and after making a change, you won't know whether the change was a good idea, a bad idea, or merely irrelevant. You can monitor the performance of Web Server in several different ways.

TABLE 1-1 Methods of Monitoring Performance

Monitoring Method	How to Enable	How to Access	Advantages and Requirements
Statistics through the Admin Console	Enabled by default	In the Admin Console, for a configuration, click the Monitor tab	Accessible when session threads are hanging. Administration Server must be running.
Statistics through individual wadm commands	Enabled by default	Through wadm commands: get-config-stats get-virtual-server-stats get-webapp-stats get-servlet-stats	Accessible when session threads are hanging. Administration Server must be running.
XML-formatted statistics (stats-xml) through a browser	Enable through Admin Console or through editing a configuration file	Through a URI	Administration Server need not be running.
XML-formatted statistics (stats-xml) through the command-line interface	Enabled by default	Through the wadm command get-stats-xml	Accessible when session threads are hanging. Administration Server must be running.
perfdump through a browser	Enable through Admin Console or through editing a configuration file	Through a URI	Administration Server need not be running.
perfdump through the command-line interface	Enabled by default	Through wadm command get-perfdump	Accessible when session threads are hanging. Administration Server must be running.
Java ES Monitoring	Enabled by default	Through the Java ES Monitoring Console	Only for Java ES installations. Administration Server must be running.

Monitoring the server does have some impact on computing resources. In general, using perfdump through the URI is the least costly, followed by using stats-xml through a URI.

Because using the Administration Server requires computing resources, the command-line interface and the Admin Console are the most costly monitoring methods.

For more information on these monitoring methods, see the following sections:

- [“About Statistics” on page 21](#)
- [“Monitoring Current Activity Using the Admin Console” on page 23](#)
- [“Monitoring Current Activity Using the CLI” on page 24](#)
- [“Monitoring Current Activity Using stats.xml” on page 27](#)
- [“Monitoring Current Activity Using perfdump” on page 29](#)
- [“Monitoring Current Activity Using the Java ES Monitoring Console” on page 35](#)

About Statistics

You can monitor many performance statistics through the Admin Console user interface, through the command-line interface, through the `stats -xml` URI, and through `perfdump`. For all these monitoring methods, the server uses the statistics it collects. None of these monitoring methods will work if statistics are not collected.

The statistics give you information at the configuration level, the server instance level, or the virtual server level. The statistics are broken up into functional areas.

For the configuration, statistics are available in the following areas:

- Requests
- Errors
- Response Time

For the server instance, statistics are available in the following areas:

- Requests
- Errors
- Response Time
- General
- Java Virtual Machine (JVM™)
- Connection Queue
- Keep Alive
- DNS
- File Cache
- Thread Pools
- Session Replication
- Session Threads, including Profiling data (available if profiling is enabled)

- Java DataBase Connectivity (JDBC™) (available if a JDBC resource is created and the connection pool is accessed)

For the virtual server, statistics are available in the following areas:

- General
- Response
- Web Applications
- Profiling Data (available if profiling is enabled)
- Servlet and Servlet Response Cache (available if the Servlet cache is enabled in `sun.web.xml`)

Some statistics default to zero if Quality of Service (QoS) is not enabled, for example, the count of open connections, the maximum open connections, the rate of bytes transmitted, and the maximum byte transmission rate.

Enabling Statistics

Statistics are activated by default on Web Server. However, if you have disabled them, you need to enable them again to monitor your server for performance. To enable statistics, use Admin Console or the `wadm` command-line utility (CLI).

Note – Collecting statistics causes a slight hit to performance.

▼ To Enable Statistics from the Admin Console

- 1 From the Admin Console Common Tasks page, select the configuration.
- 2 Click Edit Configuration.
- 3 Click the General tab.
- 4 Click the Monitoring Settings sub tab.
- 5 On the Monitoring Settings page, under General Settings, select the Statistics Enabled checkbox.
- 6 Configure the interval and profiling.
 - The Interval is the period in seconds between statistics updates. A higher setting with less frequent updates improves performance. The minimum value is .001 seconds; the default value is 5 seconds.
 - Profiling is activated by default. Deactivating it results in slightly less monitoring overhead.

- 7 Restart the server.

▼ To Enable Statistics from the CLI

- 1 Enter the following CLI command to enable statistics collection:

```
./wadm set-stats-prop --user=admin_user --password-file=password-file  
--config=myconfig enabled=true
```

To disable statistics, set `enabled` to `false`.

- 2 To set the interval and enable profiling, use the `set-stats-prop interval` and `profiling` properties. For more information, see the help for `set-stats-prop`.
- 3 Restart the server.

Monitoring Current Activity Using the Admin Console

Frequently-used statistics are available through the Admin Console, viewed as general statistics, instance statistics, and virtual server statistics.

▼ To Monitor Statistics from the Admin Console

- 1 In the Admin Console, from the Common Tasks page, select the Monitoring tab.

- 2 Select the configuration.

The configuration statistics are displayed.

- 3 From the drop-down list, select a View interval.

The statistics displayed in your browser are automatically updated at this interval.

- 4 Select the type of statistics to display.

The initial list of statistics types includes General Statistics, Instance Statistics, and Virtual Server Statistics.

If you choose Instance Statistics, click the name of the instance to monitor. Detailed statistics are then displayed, including information on processes and session replications.

If you choose Virtual Server Statistics, click the name of the virtual server to monitor. Statistics for the virtual server are displayed, including response statistics and web application statistics. This information is not provided through `perfdump`.

Monitoring Current Activity Using the CLI

You can also view statistics information using the `wadm` commands `get-config-stats`, `get-virtual-server-stats`, `get-webapp-stats` and `get-servlet-stats`. Note that the examples below do not contain all possible command options. For the complete syntax, see the help for the command.

▼ To Monitor Statistics from the CLI

- 1 To deploy statistics for configuration on a single node, enter:

```
./wadm get-config-stats --user=admin-user --password-file=admin-password-file  
--config=config-name --node=node-name
```

Using the `node` option in this syntax restricts the output to a single node. To view the statistics at the configuration level, use the command without the `node` option.

The following shows an example of the output for a single node:

```
timeStarted=1168035653  
secondsRunning=1404  
countRequests=690546  
rpsLast1MinAvg=4491.7666  
rpsLast5MinAvg=1844.6061  
rpsLast15MinAvg=637.37305  
countErrors=0  
epsLast1MinAvg=0.0  
epsLast5MinAvg=0.0  
epsLast15MinAvg=0.0  
maxResponseTime=0.30789953  
rtLast1MinAvg=5.3970284  
rtLast5MinAvg=5.208407  
rtLast15MinAvg=35.56042  
countBytesReceived=96800935  
countBytesTransmitted=689929574  
countChildDied=0  
countVirtualServers=2  
instanceName=https-test  
process.1.countThreadPools=2  
process.1.jdbcPoolCount=1  
process.1.countThreads=64  
process.1.fractionSystemMemoryUsage=2887.0  
process.1.countConnectionQueues=1  
process.1.sizeResident=0  
process.1.countIdleThreads=32  
process.1.mode=1  
process.1.sizeVirtual=0  
process.1.countConfigurations=1  
process.1.pid=15874
```



```
process.1.timeStarted=Jan 5, 2007 2:20:53 PM
process.1.DNSCache.countCacheHits=687804
process.1.DNSCache.countAsyncNameLookup=0
process.1.DNSCache.countAsyncLookupsInProgress=0
process.1.DNSCache.flagAsyncEnabled=false
process.1.DNSCache.countAsyncAddrLookups=0
process.1.DNSCache.flagCacheEnabled=true
process.1.DNSCache.countCacheMisses=75
process.1.JDBCPool.1.countQueued=32
process.1.JDBCPool.1.countFreeConnections=0
process.1.JDBCPool.1.peakConnections=32
process.1.JDBCPool.1.millisecondsPeakWait=72
process.1.JDBCPool.1.countWaitQueueTimeouts=288
process.1.JDBCPool.1.peakQueued=64
process.1.JDBCPool.1.maxConnections=32
process.1.JDBCPool.1.currentConnections=32
process.1.JDBCPool.1.millisecondsAverageQueued=1.0
process.1.JDBCPool.1.countTotalFailedValidationConnections=0
process.1.JDBCPool.1.countLeasedConnections=32
process.1.JDBCPool.1.countTotalLeasedConnections=414
process.1.JDBCPool.1.countConnectionIdleTimeouts=1
process.1.JDBCPool.1.name=jdbc/jdbc-simple_1
process.1.connectionQueue.1.countQueued15MinuteAverage=4.3203125
process.1.connectionQueue.1.countQueued=0
process.1.connectionQueue.1.countQueued1MinuteAverage=0.046875
process.1.connectionQueue.1.countTotalQueued=79171
process.1.connectionQueue.1.countQueued5MinuteAverage=4.03125
process.1.connectionQueue.1.countOverflows=0
process.1.connectionQueue.1.maxQueued=1288
process.1.connectionQueue.1.ticksTotalQueued=724956383
process.1.connectionQueue.1.countTotalConnections=863
process.1.connectionQueue.1.peakQueued=64
process.1.connectionQueue.1.name=cq1
process.1.fileCache.countContentMisses=7
process.1.fileCache.maxMmapCacheSize=0
process.1.fileCache.sizeHeapCache=27520
process.1.fileCache.countMisses=22
process.1.fileCache.countContentHits=620662
process.1.fileCache.maxEntries=1024
process.1.fileCache.flagEnabled=true
process.1.fileCache.secondsMaxAge=30
process.1.fileCache.sizeMmapCache=0
process.1.fileCache.countInfoHits=1862013
process.1.fileCache.maxHeapCacheSize=10747924
process.1.fileCache.countOpenEntries=0
process.1.fileCache.countHits=2482682
process.1.fileCache.maxOpenEntries=1024
process.1.fileCache.countEntries=12
```

```
process.1.fileCache.countInfoMisses=19
process.1.jvm.countGarbageCollections=96
process.1.jvm.sizeHeap=67762048
process.1.jvm.countThreads=79
process.1.jvm.countClassesUnloaded=0
process.1.jvm.vMVendor=Sun Microsystems Inc.
process.1.jvm.countTotalClassesLoaded=3170
process.1.jvm.vMName=Java HotSpot(TM) Server VM
process.1.jvm.countTotalThreadsStarted=81
process.1.jvm.countClassesLoaded=3170
process.1.jvm.peakThreads=79
process.1.jvm.millisecondsGarbageCollection=1981
process.1.jvm.vMVersion=1.5.0_09-b03
process.1.keepalive.countConnections=32
process.1.keepalive.maxConnections=200
process.1.keepalive.countFlushes=0
process.1.keepalive.countRefusals=0
process.1.keepalive.countTimeouts=6
process.1.keepalive.countHits=686943
process.1.keepalive.secondsTimeout=30
process.1.threadPool.1.countQueued=0
process.1.threadPool.1.countThreadsIdle=1
process.1.threadPool.1.threadPoolId=NativePool
process.1.threadPool.1.maxThreads=128
process.1.threadPool.1.countThreads=1
process.1.threadPool.1.maxQueued=0
process.1.threadPool.1.peakQueued=0
process.1.threadPool.1.name=NativePool
process.1.threadPool.2.countQueued=0
process.1.threadPool.2.countThreadsIdle=1
process.1.threadPool.2.threadPoolId=my-custom-pool
process.1.threadPool.2.maxThreads=128
process.1.threadPool.2.countThreads=1
process.1.threadPool.2.maxQueued=0
process.1.threadPool.2.peakQueued=0
process.1.threadPool.2.name=my-custom-pool
```

2 To obtain statistics for a virtual server, enter:

```
./wadm get-virtual-server-stats --user=admin-user
--password-file=admin-password-file --config=config-name --vs=virtual-server-name
```

Because the node option is not used, this syntax provides aggregate statistics for the virtual server across all the nodes where the configuration has been deployed. Using the node option restricts the output to a single node.

3 To obtain statistics for a deployed web application, enter:

```
./wadm get-webapp-stats --user=admin-user --password-file=admin-password-file
--config=config-name --node=node-name --vs=virtual-server-name --uri=URI
```

The syntax obtains statistics for a given web application deployed on the given virtual server of the given instance. To obtain the aggregated web application statistics for a given configuration across all the nodes where the configuration has been deployed, use the command without the node option.

The following example shows the output for the URI hello:

```
countActiveSessions=1
countExpiredSessions=0
countJsps=1
countRejectedSessions=0
countReloadedJsps=1
countSessions=1
peakActiveSessions=1
secondsSessionAliveAverage=0
secondsSessionAliveMax=0
uri=/hello
vsName=myvs.sun.com
```

Monitoring Current Activity Using stats.xml

You can also display statistics using `stats -xml`, which displays statistics in XML format. The output of `stats -xml` is in XML so that various tools can easily parse the statistics. You can view the `stats -xml` output through a URI, which you have to enable, or you can view the `stats -xml` output through the CLI, which is enabled by default.

▼ To Enable the stats-xml URI from the Admin Console

If you enable the `stats -xml` URI, you can access statistics for your server in XML format through a browser. Note that when you use the `stats -xml` URI, you can access statistics even when the Administration Server is not running. Also, with the `stats -xml` URI activated, users can see the statistics information for your server, unless you take precautions to deny access.

- 1 On the **Common Tasks** page, select the configuration from the pull-down menu on the left.
- 2 Select the virtual server from the pull-down menu on the right, then click **Edit Virtual Server**.
- 3 On the **Server Settings** tab, click the **Monitoring Settings** sub tab.
- 4 Select the **XML Report enabled** checkbox.
- 5 Provide a URI, for example, `/stats-xml`.
- 6 Click **Save**.
- 7 Deploy the configuration.

8 Access the stats-xml URI, for example:

`http://yourhost:port/stats-xml`

The statistics are displayed in XML format.

▼ To Enable the stats-xml URI from the CLI**1 Use the following command to enable stats-xml:**

```
./wadm enable-stats-xml --user=admin-user --password-file=admin-password-file  
[--uri-prefix=prefix] --config=config-name --vs=virtual-server-name
```

Use the uri-prefix option to set the stats-xml URI.

2 Deploy the configuration using the wadm deploy-config command.**3 Access the stats-xml URI, for example:**

`http://yourhost:port/stats-xml`

The statistics are displayed in XML format.

▼ To Limit the stats-xml Statistics Displayed in the URI

You can modify the stats-xml URI to limit the data it provides.

● Modify the stats-xml URI to limit the information by setting elements to 0 or 1. An element set to 0 is not displayed on the stats-xml output. For example:

`http://yourhost:port/stats-xml?thread=0&process=0`

This syntax limits the stats-xml output so that thread and process statistics are not included. By default all statistics are enabled (set to 1).

Most of the statistics are available at the server level, but some are available at the process level.

Use the following syntax elements to limit stats-xml:

- cache-bucket
- connection-queue
- connection-queue-bucket (process-level)
- cpu-info
- dns-bucket
- jdbc-resource-bucket
- keepalive-bucket
- process
- profile
- profile-bucket (process-level)
- request-bucket
- servlet-bucket

- session-replication
- thread
- thread-pool
- thread-pool-bucket (process-level)
- virtual-server
- web-app-bucket

▼ To View stats-xml Output from the CLI

In addition to a URI, you can also access stats-xml output through the command-line interface which is enabled by default. Unlike viewing stats-xml output through the URI, the Administration Server must be running to view stats-xml output at the command-line. However, if request processing threads are hanging in your server, and you cannot use the URI, you can still access stats-xml output through the CLI.

- To view the stats-xml output through the command-line interface, enter:

```
./wadm get-stats-xml --user=admin-user --password-file=admin-password-file  
--config=config-name --node=node-name
```

Monitoring Current Activity Using perfdump

perfdump is a Server Application Function (SAF) built into Web Server that collects various pieces of performance data from the Web Server internal statistics and displays them in ASCII text. The perfdump output does not display all the statistics available through the command-line statistics or the Admin Console, but it can still be a useful tool. For example, you can still use perfdump even if the Administration Server is not running. You can view the perfdump output through the CLI, which is enabled by default, or you can view the perfdump output through a URI, which you have to enable. If you enable the URI, you must control access to the perfdump URI, otherwise it can be visible to users.

With perfdump, the statistics are unified. Rather than monitoring a single process, statistics are multiplied by the number of processes, which gives you an accurate view of the server as a whole.

For information on tuning the information displayed by perfdump, see [“Using Monitoring Data to Tune Your Server” on page 46](#).

▼ To Enable the perfdump URI from the Admin Console

You can enable perfdump URI for a virtual server through the Admin Console.

Note – The statistics displayed by `perfdump` are for the server as a whole. If you enable `perfdump` on one virtual server, it displays statistics for the whole server, not an individual virtual server.

- 1 **From Common Tasks, select a configuration.**
- 2 **Select the virtual server and click Edit Virtual Server.**
- 3 **Click the Monitoring Settings tab.**
- 4 **Select the Plain Text Report Enabled checkbox.**
- 5 **Provide a URI for accessing the report, for example `/ .perf`.**
- 6 **Click Save.**
- 7 **Deploy the configuration.**
- 8 **To access `perfdump`, access the URI on the virtual server.**

For example: `http://localhost:80/.perf`

You can request the `perfdump` statistics and specify how frequently the browser should automatically refresh as measured in seconds. The following example sets the refresh to every 5 seconds:

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

▼ **To Enable the `perfdump` URI from the CLI**

- 1 **Use the following command to enable `perfdump`:**
`./wadm enable-perfdump --user=admin-user --password-file=admin-password-file
[--uri=uri] --config=config-name --vs=virtual-server-name`

Use the `uri` option to set the `perfdump` URI.

- 2 **Deploy the configuration using the `wadm deploy-config` command.**
- 3 **To access `perfdump`, access the URI on the virtual server.**

For example: `http://localhost:80/.perf`

You can request the `perfdump` statistics and specify how frequently the browser should automatically refresh as measured in seconds. The following example sets the refresh to every 5 seconds:

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

▼ To View the perfdump Data from the CLI

In addition to a URI, you can also access perfdump output through the command-line interface. It is enabled by default. Unlike viewing perfdump output through the URI, the Administration Server must be running to view perfdump output at the command-line. However, if request processing threads are hanging in your server and you cannot use the URI, you can still access perfdump output through the CLI.

- To view the perfdump output through the command-line interface, enter:

```
./wadm get-perfdump --user=admin-user --password-file=admin-password-file
--config=config-name --node=node-name
```

The output appears in your command window.

Sample perfdump Output

The following is sample perfdump output:

```
webservd pid: 29133
```

```
Sun Java System Web Server 7.0 B07/13/2006 17:09 (SunOS DOMESTIC)
```

```
Server started Fri Jul 14 14:34:15 2006
```

```
Process 29133 started Fri Jul 14 14:34:17 2006
```

```
ConnectionQueue:
```

```
-----
Current/Peak/Limit Queue Length      2/237/1352
Total Connections Queued              67364017
Average Queue Length (1, 5, 15 minutes) 4.52, 4.73, 4.85
Average Queueing Delay                13.63 milliseconds
```

```
ListenSocket ls1:
```

```
-----
Address                https://0.0.0.0:2014
Acceptor Threads       1
Default Virtual Server https-test
```

```
KeepAliveInfo:
```

```
-----
KeepAliveCount      198/200
KeepAliveHits        0
KeepAliveFlushes     0
KeepAliveRefusals    56844280
KeepAliveTimeouts    365589
KeepAliveTimeout     10 seconds
```

```
SessionCreationInfo:
```

Active Sessions 128
Keep-Alive Sessions 0
Total Sessions Created 128/128

Server cache disabled

Native pools:

NativePool:
Idle/Peak/Limit 1/1/128
Work Queue Length/Peak/Limit 0/0/0
TestPool:
Idle/Peak/Limit 5/5/10
Work Queue Length/Peak/Limit 0/0/15

DNSSCacheInfo:

enabled yes
CacheEntries 4/1024
HitRatio 62854802/62862912 (99.99%)

Async DNS disabled

Performance Counters:

	Average	Total	Percent
Total number of requests:		62647125	
Request processing time:	0.0343	2147687.2500	
default-bucket (Default bucket)			
Number of Requests:		62647125	(100.00%)
Number of Invocations:		3374170785	(100.00%)
Latency:	0.0008	47998.2500	(2.23%)
Function Processing Time:	0.0335	2099689.0000	(97.77%)
Total Response Time:	0.0343	2147687.2500	(100.00%)

Sessions:

Process	Status	Client	Age	VS	Method	URI	Function
29133	response	192.6.7.7	115	https-test	GET	/qa_webapp/CheckNetwork.class	service-j2ee
29133	response	192.6.7.7	8	https-test	GET	/qa_webapp/CheckNetwork.class	service-j2ee
29133	response	192.6.7.7	4	https-test	GET	/qa_webapp/CheckNetwork.class	service-j2ee
29133	response	10.5.8.19	4	https-test	GET	/perf	service-dump
29133	response	192.6.7.7	3	https-test	GET	/qa_webapp/CheckNetwork.class	service-j2ee
29133	response	192.6.7.7	3	https-test	GET	/qa_webapp/CheckNetwork.class	service-j2ee

```

29133 response 192.6.7.7 2 https-test GET /qa_webapp/CheckNetwork.class service-j2ee
29133 response 192.6.7.7 2 https-test GET /qa_webapp/CheckNetwork.class service-j2ee
29133 response 192.6.7.7 2 https-test GET /qa_webapp/CheckNetwork.class service-j2ee
29133 response 192.6.7.7 2 https-test GET /qa_webapp/CheckNetwork.class service-j2ee
29133 request 192.6.7.7 0
29133 request 192.6.7.7 0
29133 request 192.6.7.7 0
29133 request 192.6.7.7 0
29133 response 192.6.7.7 0 https-test GET /file1.shtml shtml_send
29133 request 192.6.7.7 0
29133 request 192.6.7.7 0
29133 response 192.6.7.7 0 https-test GET /find-pathinfo-forward/pathinfo.pl/p/info send-cgi
29133 request 192.6.7.7 0
29133 updating 192.6.7.7
29133 updating 192.6.7.7
29133 updating 192.6.7.7
29133 updating 192.6.7.7
.
.
.

```

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 `service-j2ee` 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 minimal, and the impact on the server performance is usually negligible. This information can later be accessed using `perfdump`. The following information is stored in a bucket:

- **Name of the bucket.** This name associates the bucket with a function.
- **Description.** A description of the functions with which the bucket is associated.
- **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 might not coincide with the number of requests for the function, because some functions might 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 predefined by the server. It records statistics for the functions not associated with any user-defined bucket.

Configuration

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

First, you must enable performance statistics collection and `perfdump`.

The following examples show how to define new buckets in `magnus.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 previous examples create three buckets: `acl-bucket`, `file-bucket`, and `cgi-bucket`. To associate these buckets with functions, add `bucket=`*bucket-name* to the `obj.conf` function for which to measure performance.

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>
```

For more information, see [“The bucket Parameter” in Sun Java System Web Server 7.0 Update 8 Administrator’s Configuration File Reference](#).

Performance Report

The server statistics in buckets can be accessed using `perfdump`. The performance buckets information is located in the last section of the report returned by `perfdump`.

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 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 number differs from the number of requests because 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 of the buckets.
- Latency is the time in seconds that Web Server uses to call the function.
- Function Processing Time is the time in seconds that Web Server spends 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:		62647125	
Request processing time:	0.0343	2147687.2500	
default-bucket (Default bucket)			
Number of Requests:		62647125	(100.00%)
Number of Invocations:		3374170785	(100.00%)
Latency:	0.0008	47998.2500	(2.23%)
Function Processing Time:	0.0335	2099689.0000	(97.77%)
Total Response Time:	0.0343	2147687.2500	(100.00%)

Monitoring Current Activity Using the Java ES Monitoring Console

The statistics available through the Web Server Admin Console and the command-line interface are also available through the Java ES Monitoring Console. Though the information is the same, it is presented in a different format, using the Common Monitoring Data Model (CMM). Though this guide covers monitoring using tools available in the Web Server, you can also monitor your server using the Java ES monitoring tools. For more information on using the Java ES monitoring tools, see *Sun Java Enterprise System 5 Monitoring Guide*. Use the same settings to tune the server, regardless of what monitoring method you use.

Tuning Sun Java System Web Server

This chapter describes specific adjustments you can make that might improve Sun Java System Web Server performance. It provides an overview of Web Server's connection-handling process so that you can better understand the tuning settings. The chapter includes the following topics:

- “General Tuning Tips” on page 37
- “Understanding Threads, Processes, and Connections” on page 38
- “Mapping Web Server 6.1 Tuning Parameters to Web Server 7.0” on page 44
- “Using Monitoring Data to Tune Your Server” on page 46
- “Tuning the ACL User Cache” on page 77
- “Tuning Java Web Application Performance” on page 78
- “Tuning CGI Stub Processes (UNIX/Linux)” on page 81
- “Using find-pathinfo-forward” on page 81
- “Using nostat” on page 82
- “Using Busy Functions” on page 83

Note – Be very careful when tuning your server. Always back up your configuration files before making any changes.

General Tuning Tips

As you tune your server, it is important to remember that your specific environment is unique. The impacts of the suggestions provided in this guide will vary, depending on your specific environment. Ultimately you must rely on your own judgement and observations to select the adjustments that are best for you.

As you work to optimize performance, keep the following guidelines in mind:

- **Work methodically**

As much as possible, make one adjustment at a time. Measure your performance before and after each change, and rescind any change that doesn't produce a measurable improvement.

- **Adjust gradually**

When adjusting a quantitative parameter, make several changes in succession, rather than trying to make a drastic change all at once. Different systems face different circumstances, and you might pass by your system's best setting if you change the value too rapidly.

- **Start fresh**

At each major system change, be it a hardware or software upgrade or deployment of a major new application, review all previous adjustments to see whether they still apply. After a Solaris upgrade, you should start over with an unmodified `/etc/system` file.

- **Stay informed**

Read the [Sun Java System Web Server 7.0 Update 8 Release Notes](#) and the release notes for your operating system whenever you upgrade your system. The release notes often provide updated information about specific adjustments.

Understanding Threads, Processes, and Connections

Before tuning your server, you should understand the connection-handling process in Web Server. Request processing threads handle Web Server connections. You can configure Request handling threads from the Admin Console or by editing the `server.xml` configuration file. This section includes the following topics:

- [“Connection-Handling Overview” on page 38](#)
- [“Custom Thread Pools” on page 40](#)
- [“The Native Thread Pool” on page 41](#)
- [“Process Modes” on page 42](#)

Connection-Handling Overview

In Web Server, acceptor threads on a listen socket accept connections and put them into a connection queue. Request processing threads in a thread pool then pick up connections from the queue and service the requests.

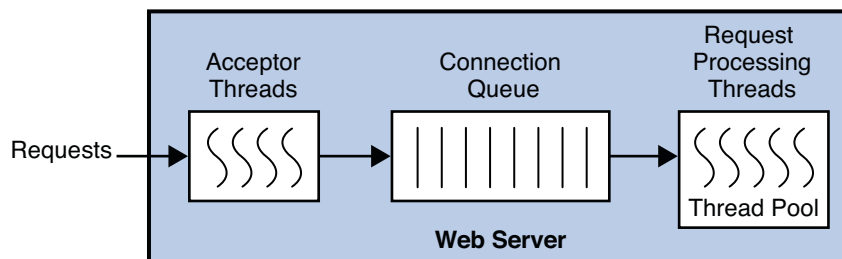


FIGURE 2-1 Web Server Connection Handling

A request is not thread-safe if processing the request requires interaction between a number of threads. A part of the request which is not thread-safe is transferred to a NativePool, which is a collection of threads which can interact with each other. The NativePool processes the request and communicates the request back to the request processing thread.

At startup, the server only creates the number of threads defined in the thread pool minimum threads, by default set to number of processors. As the load increases, the server creates more threads. The policy for adding new threads is based on the connection queue state.

Each time a new request is created, the number of requests waiting in the queue, often considered the backlog of connections, is compared to the number of request processing threads already created. If the number of requests is greater than the number of threads, more threads are created.

The process of adding new session threads is strictly limited by the maximum threads value. For more information on maximum threads, see [“Maximum Threads \(Maximum Simultaneous Requests\)”](#) on page 57.

You can change the settings that affect the number and timeout of threads, processes, and connections in the Admin Console, on the configuration's Performance tab (HTTP settings), and on the HTTP listener. You can also use the wadm commands `set-thread-pool-prop` and `set-http-listener-prop` and `set-keep-alive-prop`.

Low Latency and High Concurrency Modes

The server can run in one of two modes, depending upon the load. It changes modes to accommodate the load most efficiently.

- In low latency mode, for keep-alive connections, session threads themselves poll for new requests.
- In high concurrency mode, after finishing the request, session threads give the connection to the keep-alive subsystem. In high concurrency mode, the keep-alive subsystem polls for new requests for all keep-alive connections.

When the server is started, it starts in low latency mode. When the load increases, the server moves to high concurrency mode. The decision to move from low latency mode to high concurrency mode and back again is made by the server, based on connection queue length, average total sessions, average idle sessions, and currently active and idle sessions.

Disabled Thread Pools

If a thread pool is disabled, no threads are created in the pool, no connection queue is created, and no keep-alive threads are created. When the thread pool is disabled, the acceptor threads themselves process the request.

Connection–Handling `magnus.conf` Directives for NSAPI

In addition to the settings discussed above, you can edit the following directives in the `magnus.conf` file to configure additional request-processing settings for NSAPI plug-ins:

- `KernelThreads` – Determines whether NSAPI plug-ins always run on kernel-scheduled threads (Windows only)
- `TerminateTimeout` – Determines the maximum amount of time to wait for NSAPI plug-ins to finish processing requests when the server is shut down

For detailed information about these directives, see the [Sun Java System Web Server 7.0 Update 8 Administrator's Configuration File Reference](#).

Note – For the safest way to edit configuration files such as `magnus.conf`, use the `wadm` commands `get-config-file` and `set-config-file` to pull a local copy for editing and push it back to the Web Server. For more information on these commands, see the help for these commands.

Custom Thread Pools

By default, the connection queue sends requests to the default thread pool. However, you can also create your own thread pools in `magnus.conf` using a `thread pool Init` function. These custom thread pools are used for executing NSAPI Service Application Functions (SAFs), not entire requests.

If the SAF requires the use of a custom thread pool, the current request processing thread queues the request, waits until the other thread from the custom thread pool completes the SAF, then the request processing thread completes the rest of the request.

For example, the `obj.conf` file contains the following:


```

NameTrans fn="assign-name" from="/testmod" name="testmod" pool="my-custom-pool"
...
<Object name="testmod">
ObjectType fn="force-type" type="magnus-internal/testmod"
Service method=(GET|HEAD|POST) type="magnus-internal/testmod"
fn="testmod_service" pool="my-custom-pool2"
</Object>

```

In this example, the request is processed as follows:

1. The request processing thread, referred to as A1 in this example, picks up the request and executes the steps before the NameTrans directive.
2. If the URI starts with /testmod, the A1 thread queues the request to the my-custom-pool queue. The A1 thread waits.
3. A different thread in my-custom-pool, called the B1 thread in this example, picks up the request queued by A1. B1 completes the request and returns to the wait stage.
4. The A1 thread wakes up and continues processing the request. It executes the ObjectType SAF and moves on to the Service function.
5. Because the Service function must be processed by a thread in my-custom-pool2, the A1 thread queues the request to my-custom-pool2.
6. A different thread in my-custom-pool2, called C1 in this example, picks up the queued request. C1 completes the request and returns to the wait stage.
7. The A1 thread wakes up and continues processing the request.

In this example, three threads, A1, B1, and C1 work to complete the request.

Additional thread pools are a way to run thread-unsafe plug-ins. By defining a pool with a maximum number of threads set to 1, only one request is allowed into the specified service function. In the previous example, if testmod_service is not thread-safe, it must be executed by a single thread. If you create a single thread in the my-custom-pool2, the SAF works in a multi-threaded Web Server.

For more information on defining thread pools, see [“thread-pool-init” in Sun Java System Web Server 7.0 Update 8 Administrator’s Configuration File Reference](#).

The Native Thread Pool

On Windows, the native thread pool (NativePool) is used internally by the server to execute NSAPI functions that require a native thread for execution.

Web Server uses Netscape Portable Runtime (NSPR), which is an underlying portability layer providing access to the host OS services. This layer provides abstractions for threads that are not always the same as those for the OS-provided threads. These non-native threads have lower scheduling overhead, so their use improves performance. However, these threads are sensitive

to blocking calls to the OS, such as I/O calls. To make it easier to write NSAPI extensions that can make use of blocking calls, the server keeps a pool of threads that safely support blocking calls. These threads are usually native OS threads. During request processing, any NSAPI function that is not marked as being safe for execution on a non-native thread is scheduled for execution on one of the threads in the native thread pool.

If you have written your own NSAPI plug-ins such as NameTrans, Service, or PathCheck functions, these execute by default on a thread from the native thread pool. If your plug-in makes use of the NSAPI functions for I/O exclusively or does not use the NSAPI I/O functions at all, then it can execute on a non-native thread. For this to happen, the function must be loaded with a `NativeThread="no"` option, indicating that it does not require a native thread.

For example, add the following to the `load-modules Init` line in the `magnus.conf` file:

```
Init funcs="pcheck_uri_clean_fixed_init" shlib="C:/Sun/webserver7/lib/custom.dll"  
fn="load-modules" NativeThread="no"
```

The `NativeThread` flag affects all functions in the `funcs` list, so if you have more than one function in a library, but only some of them use native threads, use separate `Init` lines. If you set `NativeThread` to yes, the thread maps directly to an OS thread.

For information on the `load-modules` function, see [“load-modules” in Sun Java System Web Server 7.0 Update 8 Administrator’s Configuration File Reference](#).

Process Modes

You can run Sun Java System Web Server in one of the following modes:

- [“Single-Process Mode” on page 42](#)
- [“Multi-Process Mode” on page 43](#)

Note – Multi-process mode is deprecated for Java technology-enabled servers. Most applications are now multi-threaded, and multi-process mode is usually not needed. However, multi-process mode can significantly improve overall server throughput for NSAPI applications that do not implement fine-grained locking.

Single-Process Mode

In the single-process mode, the server receives requests from web clients to a single process. Inside the single server process, acceptor threads are running that are waiting for new requests to arrive. When a request arrives, an acceptor thread accepts the connection and puts the request into the connection queue. A request processing thread picks up the request from the connection queue and handles the request.

Because the server is multi-threaded, all NSAPI extensions written to the server must be thread-safe. This means that if the NSAPI extension uses a global resource, like a shared reference to a file or global variable, then the use of that resource must be synchronized so that only one thread accesses it at a time. All plug-ins provided with the Web Server are thread-safe and thread-aware, providing good scalability and concurrency. However, your legacy applications might be single-threaded. When the server runs the application, it can only execute one at a time. This leads to server performance problems when put under load. Unfortunately, in the single-process design, there is no real workaround.

Multi-Process Mode

You can configure the server to handle requests using multiple processes with multiple threads in each process. This flexibility provides optimal performance for sites using threads, and also provides backward compatibility to sites running legacy applications that are not ready to run in a threaded environment. Because applications on Windows generally already take advantage of multi-thread considerations, this feature applies to UNIX and Linux platforms.

The advantage of multiple processes is that legacy applications that are not thread-aware or thread-safe can be run more effectively in Sun Java System Web Server. However, because all of the Sun Java System extensions are built to support a single-process threaded environment, they might not run in the multi-process mode. The Search plug-ins fail on startup if the server is in multi-process mode, and if session replication is enabled, the server will fail to start in multi-process mode.

In the multi-process mode, the server spawns multiple server processes at startup. Depending on the configuration, each process contains one or more threads, that receive incoming requests. Since each process is completely independent, each one has its own copies of global variables, caches, and other resources. Using multiple processes requires more resources from your system. Also, if you try to install an application that requires shared state, it has to synchronize that state across multiple processes. NSAPI provides no helper functions for implementing cross-process synchronization.

When you specify a `MaxProcs` value greater than 1, the server relies on the operating system to distribute connections among multiple server processes (see [“MaxProcs \(UNIX/Linux\)” on page 44](#) for information about the `MaxProcs` directive). However, many modern operating systems do not distribute connections evenly, particularly when there are a small number of concurrent connections.

Because Sun Java System Web Server cannot guarantee that load is distributed evenly among server processes, you might encounter performance problems if you set `Maximum Threads` to 1 and `MaxProcs` greater than 1 to accommodate a legacy application that is not thread-safe. The problem is especially pronounced if the legacy application takes a long time to respond to requests, for example, when the legacy application contacts a back-end database. In this scenario, it might be preferable to use the default value for `Maximum Threads` and `serialize`

access to the legacy application using thread pools. For more information about creating a thread pool, see “[thread-pool-init](#)” in *Sun Java System Web Server 7.0 Update 8 Administrator’s Configuration File Reference*.

If you are not running any NSAPI in your server, you should use the default settings: one process and many threads. If you are running an application that is not scalable in a threaded environment, you should use a few processes and many threads, for example, 4 or 8 processes and 128 or 512 threads per process.

MaxProcs (UNIX/Linux)

To run a UNIX or Linux server in multi-process mode, set the MaxProcs directive to a value that is greater than 1. Multi-process mode might provide higher scalability on multi-processor machines and improve the overall server throughput on large systems such as the Sun Fire™ T2000 server. If you set the value to less than 1, it is ignored and the default value of 1 is used.

Use the MaxProcs directive to improve overall server throughput for the following types of applications:

- NSAPI applications that do not implement fine-grained locking
- Java applications that do not require session management

Do *not* use the MaxProcs directive when the Sun Java System Web Server performs session management for Java applications.

You can set the value for MaxProcs by editing the MaxProcs parameter in `magnus.conf`.

Note – You will receive duplicate startup messages when running your server in MaxProcs mode.

Mapping Web Server 6.1 Tuning Parameters to Web Server 7.0

Many of the tuning parameters that were tunable by editing the `magnus.conf` and `nsfc.conf` files in Web Server 6.1 have moved to the `server.xml` file. These tuning parameters are now tunable through the Admin Console and command-line interface. The following table shows selected tuning parameters, including the Web Server 6.1 parameter, the new `server.xml` element used for tuning, and the way to change the parameters through the user interface. Editing the `server.xml` file directly can be error-prone, so using the user interface to set values is preferable. For a complete list of all elements in `server.xml`, see [Chapter 3, “Elements in server.xml,”](#) in *Sun Java System Web Server 7.0 Update 8 Administrator’s Configuration File Reference*.

TABLE 2-1 Parameter Mapping to server.xml

Web Server 6.1 parameter	Web Server 7.0 server.xml element or attribute	Admin Console Location	wadm command
AcceptTimeout in magnus.conf	io-timeout element of the http element	Configuration's Performance tab ⇒ HTTP Settings page	set-http-prop command's io-timeout property
ACLGroupCacheSize in magnus.conf	max-groups-per-user element of the acl-cache element	Configuration's Performance tab ⇒ Cache Settings page	set-acl-cache-prop command's max-groups-per-user property
ACLUserCacheSize in magnus.conf	max-users element of the acl-cache element	Configuration's Performance tab ⇒ Cache Settings page	set-acl-cache-prop command's max-users property
ConnQueueSize in magnus.conf	queue-size element of the thread-pool element	Configuration's Performance tab ⇒ HTTP tab	set-thread-pool-prop command's queue-size property
dns-cache-init Init SAF	enabled element of the dns-cache element	Configuration's Performance tab ⇒ DNS tab	set-dns-cache-prop command's enabled property
dns-cache-init Init SAF cache size	max-entries element of the dns-cache element	Configuration's Performance tab ⇒ DNS tab	set-dns-cache-prop command's max-entries property
FileCacheEnabled in nsfc.conf	enabled element of the file-cache element	Configuration's Performance tab ⇒ Cache tab	set-file-cache-prop command's enabled property
KeepAliveThreads in magnus.conf	threads element of the keep-alive element	Configuration's Performance tab ⇒ HTTP tab	set-keep-alive-prop command's threads property
KeepAliveTimeout in magnus.conf	timeout element of the keep-alive element	Configuration's Performance tab ⇒ HTTP tab	set-keep-alive-prop command's timeout property
KernelThreads in magnus.conf (Windows only)	Unchanged		
ListenQ in magnus.conf	listen-queue-size element of the http-listener element	Configuration's HTTP Listeners tab	set-http-listener-prop command's listen-queue-size
LogVerbose in magnus.conf	log-level element of the log element	Configuration's General Tab ⇒ Log Settings	set-error-log-prop command's log-level property

TABLE 2-1 Parameter Mapping to server.xml (Continued)

Web Server 6.1 parameter	Web Server 7.0 server.xml element or attribute	Admin Console Location	wadm command
MaxAge in nsfc.conf file	max-age element of the file-cache element	Configuration's Performance tab ⇒ Cache tab	set-file-cache-prop command's max-age property
MaxFiles in nsfc.conf file	max-entries element of the file-cache element	Configuration's Performance tab ⇒ Cache tab	set-file-cache-prop command's max-entries property
MaxKeepAliveConnections in magnus.conf	max-connections element of the keep-alive element	Configuration's Performance tab ⇒ HTTP tab	set-keep-alive-prop command's max-connections property
MaxProcs in magnus.conf	Deprecated for Java technology-enabled servers		
NativePoolMaxThreads in magnus.conf	Unchanged		
NativePoolMinThreads in magnus.conf	Unchanged		
NativePoolQueueSize in magnus.conf	Unchanged		
NativePoolStackSize in magnus.conf	Unchanged		
RqThrottle in magnus.conf	max-threads element of the thread-pool element	Configuration's Performance tab ⇒ HTTP tab	set-thread-pool-prop command's max-threads property
RqThrottleMin in magnus.conf	min-threads element of the thread-pool element	Configuration's Performance tab ⇒ HTTP tab	set-thread-pool-prop command's min-threads property
TerminateTimeout in magnus.conf	Unchanged		

Using Monitoring Data to Tune Your Server

This section describes the performance information available through the Admin Console, perfdump, the command-line interface, and stats-xml. It discusses how to analyze that information and tune parameters to improve your server's performance.

Web Server automatically selects many server defaults based on the system resources. The number of acceptor threads and keep-alive threads defaults to the number of CPUs. The

`server/thread-pool/max-threads` defaults to greater of 128 or the number of CPUs. The `server/thread-pool/min-threads` defaults to lesser the value of `server/thread-pool/max-threads` or the number of CPUs. The `server/access-log-buffer/max-buffers-per-file` defaults to the number of CPUs. The server configures the connection queue size, maximum number of keep-alive connections, and the maximum number of open files in the file cache, based on the total number of available file descriptors in the system. The values for these are obtained from the server log file when the log level is set to fine. All the server chosen defaults are tunable.

The default tuning parameters are appropriate for all sites except those with very high volume. The only settings that large sites might regularly need to change are the thread pool and keep alive settings. Tune these settings at the configuration level in the Admin Console or using `wadm` commands. It is also possible to tune the server by editing the elements directly in the `server.xml` file, but editing the `server.xml` file directly can lead to complications.

`perfdump` monitors statistics in the following categories, which are described in the following sections. In most cases these statistics are also displayed in the Admin Console, command-line interface, and `stats-xml` output. The following sections contain tuning information for all these categories, regardless of which method you use to monitor the data:

- [“Connection Queue Information” on page 48](#)
- [“HTTP Listener \(Listen Socket\) Information” on page 50](#)
- [“Keep-Alive Information” on page 51](#)
- [“Session Creation and Thread Information” on page 56](#)
- [“File Cache Statistics Information” on page 58](#)
- [“Thread Pool Information” on page 65](#)
- [“DNS Cache Information” on page 68](#)

In addition, the statistics information displayed through the Admin Console, the command-line interface, and `stats-xml` contains other categories not contained in the `perfdump` output. Tuning these statistics is discussed in the following sections:

- [“Java Virtual Machine \(JVM\) Information” on page 70](#)
- [“Web Application Information” on page 71](#)
- [“JDBC Resource Information” on page 72](#)

Once you have viewed the statistics you need, you can tune various aspects of your server’s performance at the configuration level using the Admin Console's Performance tab. The Admin Console Performance tab includes settings for many performance categories, including:

- HTTP Settings (includes Thread Pool and Keep Alive)
- DNS Settings
- SSL and TLS Settings
- Cache Settings
- CGI Settings
- Access Log Buffer Settings

You can also view and set tuning parameters using the appropriate `wadm` commands. When you set tuning properties using `wadm` commands, the names of the properties are the same as displayed in `stats.xml`.

Connection Queue Information

In Web Server, a connection is first accepted by acceptor threads associated with the HTTP listener. The acceptor threads accept the connection and put it into the connection queue. Then, request processing threads take the connection in the connection queue and process the request. For more information, see [“Connection-Handling Overview” on page 38](#).

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

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

```
ConnectionQueue:
-----
Current/Peak/Limit Queue Length      0/1853/160032
Total Connections Queued              11222922
Average Queue Length (1, 5, 15 minutes)  90.35, 89.64, 54.02
Average Queueing Delay                4.80 milliseconds
```

The same information is displayed in a different format through the Admin Console or command-line interface, with some slight differences. The following table shows the information displayed in the Admin Console when accessing monitoring information for the server instance:

TABLE 2-2 Connection Queue Statistics

Present Number of Connections Queued	0
Total Number of Connections Queued	11222922
Average Connections Over Last 1 Minute	90.35
Average Connections Over Last 5 Minutes	89.64
Average Connections Over Last 15 Minutes	54.02
Maximum Queue Size	160032
Peak Queue Size	1853
Number of Connections Overflowed	0
Ticks Spent	5389284274
Total Number of Connections Added	425723

Current /Peak /Limit Queue Length

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. This number is:

$$\text{Maximum Queue Size} = \text{Thread Pool Queue Size} + \text{Maximum Threads} + \text{Keep-Alive Queue Size}$$

Once the connection queue is full, new connections are dropped.

Tuning

If the peak queue length, also known as the maximum queue size, is close to the limit, you can increase the maximum connection queue size to avoid dropping connections under heavy load.

You can increase the maximum connection queue size in the Admin Console by changing the value of the thread pool Queue Size field on the configuration's Performance tab ⇒ HTTP sub tab. The default is selected based on the total number of available file descriptors in the system.

To change the queue size using the command-line interface, use the `wadm set-thread-pool-prop` command's `queue-size` property.

Total Connections Queued

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

This setting is not tunable.

Average Queue Length

The Average Queue Length shows the average number of connections in the queue over the most recent one-minute, five-minute, and 15-minute intervals.

This setting is not tunable.

Average Queuing Delay

The Average Queuing 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 when a request processing thread begins servicing the request. It is the Ticks Spent divided by the Total Connections Queued, and converted to milliseconds.

This setting is not tunable.

Ticks Spent

A tick is a system-dependent value and provided by the `tickPerSecond` attribute of the `server` element in `stats.xml`. The ticks spent value is the total amount of time that connections spent in the connection queue and is used to calculate the average queuing delay.

This setting is not tunable.

Total Number of Connections Added

The new connections added to the connection queue. This setting is not tunable.

HTTP Listener (Listen Socket) Information

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

You can have many HTTP listeners enabled for virtual servers, but at least one is enabled for your default server instance (usually `http://0.0.0.0:80`). The monitoring information available through the Admin Console does not show the HTTP listener information, because that information is available in the Admin Console on the configuration's HTTP Listeners tab.

The following is an example of how the HTTP listeners information appears in `perfdump`:

```
ListenSocket ls1:
-----
Address           https://0.0.0.0:2014
Acceptor Threads   1
Default Virtual Server https-test
```

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

To edit an HTTP listener using the Admin Console, for the configuration, select the HTTP Listeners tab. Click the listener name to edit the listener.

To configure an HTTP listener using the command-line interface, use the command `wadm set-http-listener-prop`.

For more information about adding and editing listen sockets, see the [Sun Java System Web Server 7.0 Update 8 Administrator's Guide](#).

Address

The Address field contains the base address on which this listen socket is listening. A host can have multiple network interfaces and multiple IP addresses. The address contains the IP address and the port number.

If your listen socket listens on all network interfaces for the host machine, the IP part of the address is 0.0.0.0.

Tuning

This setting is tunable when you edit an HTTP listener. If you specify an IP address other than 0.0.0.0, the server makes one less system call per connection. Specify an IP address other than 0.0.0.0 for 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. For more information, see [“Connection-Handling Overview” on page 38](#).

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 burden 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

This setting is tunable when you edit an HTTP listener. The number of acceptor threads defaults to the number of CPUs on your system.

Other HTTP listener settings that affect performance are the size of the send buffer and receive buffer. For more information regarding these buffers, see your operating system documentation.

Default Virtual Server

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, Web Server handles the request using a default virtual server. You can configure the default virtual server to send an error message or serve pages from a special document root.

Tuning

This setting is tunable when you edit an HTTP listener.

Keep-Alive Information

This section provides information about the server’s HTTP-level keep-alive system.

Note – The name keep alive should not be confused with TCP keep-alives. Also, note that the name keep-alive was changed to PersistentConnections in HTTP 1.1, but Web Server continues to refer to these connections as keep-alive connections. Most modern browsers request a web page from the server through persistent connections with the web server. The connection is kept alive even after processing a request, so that it will be easier to process a similar request.

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

```
KeepAliveInfo:
-----
KeepAliveCount      198/200
KeepAliveHits       0
KeepAliveFlushes    0
KeepAliveRefusals   56844280
KeepAliveTimeouts   365589
KeepAliveTimeout    10 seconds
```

The following table shows the keep-alive statistics displayed in the Admin Console:

TABLE 2-3 Keep-Alive Statistics

Number of Connections Processed	0
Total Number of Connections Added	198
Maximum Connection Size	200
Number of Connections Flushed	0
Number of Connections Refused	56844280
Number of Idle Connections Closed	365589
Connection Timeout	10

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 is allowed to keep the connection open indefinitely, the server can become overloaded with connections. On UNIX and Linux systems, this can lead to a file table overflow very easily.

To resolve this problem, the server maintains a counter for the maximum number of waiting keep-alive connections. A waiting keep-alive connection has fully completed processing the previous request, and is now 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.

Sun Java System Web 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:

- The keep alive timeout is set to 0.
- The keep alive maximum connections 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`).
- The 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.

The keep-alive subsystem in Web Server is designed to be massively scalable. The out-of-the-box configuration can be less than optimal if the workload is non-persistent (that is, HTTP 1.0 without the `KeepAlive` header), or for a lightly loaded system that's primarily servicing keep-alive connections.

Keep-Alive Count

This section in `perfdump` has two numbers:

- Number of connections in keep-alive mode, also known as the total number of connections added
- Maximum number of connections allowed in keep-alive mode simultaneously, also known as the maximum connection size

Tuning

You can tune the maximum number of connections that the server allows to wait at one time before closing the oldest connection in the Admin Console by editing the Maximum Connections field on the configuration's Performance tab ⇒ HTTP tab, under Keep Alive Settings. The default is based on the number of available file descriptors in the system. In the command-line interface, use the `max-connections` property in the `wadm set-keep-alive-prop` command.

Note – The number of connections specified by the maximum connections setting is divided equally among the keep-alive threads. If the maximum connections setting is not equally divisible by the keep-alive threads setting, the server might allow slightly more than the maximum number of simultaneous keep-alive connections.

Keep-Alive Hits

The keep-alive hits, or the number of connections processed, is the number of times a request was successfully received from a connection that was kept alive.

This setting is not tunable.

Keep-Alive Flushes

The number of times the server had to close a connection because the total number of connections added exceeded the keep-alive maximum connections setting. The server does not close existing connections when the keep-alive count exceeds the maximum connection size. Instead, new keep-alive connections are refused and the number of connections refused count is incremented.

Keep-Alive Refusals

The number of times the server could not complete the connection to a keep-alive thread, possibly due to too many persistent connections (or when total number of connections added exceeds the keep-alive maximum connections setting). The suggested tuning is to increase the keep-alive maximum connections.

Keep-Alive Timeouts

The number of times the server closed idle keep-alive connections because client connections timed out without any activity. This statistic is useful to monitor; no specific tuning is advised.

Keep-Alive Timeout

The time, measured in seconds, before idle keep-alive connections are closed. Set this value in the Admin Console in the Timeout field on the configuration's Performance tab ⇒ HTTP tab, under Keep Alive Settings. The default is 30 seconds, meaning the connection times out if idle for more than 30 seconds. The maximum is 3600 seconds (60 minutes). In the command-line interface, use the `timeout` property in the `wadm set-keep-alive-prop` command.

Keep-Alive Poll Interval

The keep-alive poll interval specifies the interval in seconds at which the system polls keep-alive connections for further requests. The default is 0.001 second, the lowest value allowed. It is set to a low value to enhance performance at the cost of CPU usage.

To tune the poll interval, edit the Poll Interval field on the configuration's Performance tab ⇒ HTTP tab, under Keep Alive Settings. In the command-line interface, use the `poll-interval` property in the `wadm set-keep-alive-prop` command.

Keep-Alive Threads

You can configure the number of threads used in the keep-alive system in the Admin Console by editing the Threads field on the configuration's Performance tab ⇒ HTTP tab, under Keep Alive Settings. The default is set to the number of processors in the system. In the command-line interface, use the threads property in the `wadm set-keep-alive-prop` command.

Tuning for HTTP 1.0-Style Workload

Since HTTP 1.0 results in a large number of new incoming connections, the default acceptor threads of 1 per listen socket would be suboptimal. Increasing this to a higher number should improve performance for HTTP 1.0-style workloads. For instance, for a system with 2 CPUs, you might want to set it to 2. You might also want to reduce the keep-alive connections, for example, to 0.

HTTP 1.0-style workloads can have many connections established and terminated.

If users are experiencing connection timeouts from a browser to Web Server when the server is heavily loaded, you can increase the size of the HTTP listener backlog queue by setting the HTTP listener listen queue size to a larger value, such as 8192.

The HTTP listener listen queue specifies the maximum number of pending connections on a listen socket. Connections that time out on a listen socket whose backlog queue is full fail.

Tuning for HTTP 1.1-Style Workload

While tuning server-persistent connection handling, balancing throughput and latency is a challenge. The keep-alive poll interval and timeout control latency. Lowering the value of these settings is intended to lower latency on lightly loaded systems, for example, to reduce page load times. Increasing the values of these settings is intended to raise aggregate throughput on heavily loaded systems, for example, by increasing the number of requests per second the server can handle. However, if there's too much latency and too few clients, aggregate throughput suffers as the server sits idle unnecessarily. As a result, the general keep-alive subsystem tuning rules at a particular load are as follows:

- If there's idle CPU time, decrease the poll interval.
- If there's no idle CPU time, increase the poll interval.

Also, chunked encoding could affect the performance for HTTP 1.1 workload. Tuning the response buffer size can positively affect the performance. A higher response buffer size in the configuration's Performance tab ⇒ HTTP tab would result in sending a `Content-length:` header, instead of chunking the response. To set the buffer size using the CLI, use the `wadm set-http-prop` command's `output-buffer-size` property.

You can also set the buffer size for a Service-class function in the `obj.conf` file, using the `UseOutputStreamSize` parameter. `UseOutputStreamSize` overrides the value set using the

output-buffer-size property. If `UseOutputStreamSize` is not set, Web Server uses the output-buffer-size setting. If the output-buffer-size is not set, Web Server uses the output-buffer-size default value of 8192.

The following example shows using the CLI to increase the output buffer size, then deploying the configuration (used if `UseOutputStreamSize` is not specified in `obj.conf`):

```
./wadm set-http-prop --user=admin-user --password-file=admin-password-file
--config=config-name output-buffer-size=16384
./wadm deploy-config --user=admin-user --password-file=admin-password-file
--config=config-name
```

The following example shows setting the buffer size for the `nsapi_test` Service function:

```
<Object name="nsapitest">
  ObjectType fn="force-type" type="magnus-internal/nsapitest"
  Service method=(GET) type="magnus-internal/nsapitest" fn="nsapi_test"
  UseOutputStreamSize=12288
</Object>
```

Session Creation and Thread Information

Session (thread) creation statistics are displayed in `perfdump` as follows:

```
SessionCreationInfo:
-----
Active Sessions           128
Keep-Alive Sessions       0
Total Sessions Created    128/128
```

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

Keep-Alive Sessions shows the number of HTTP request processing threads serving keep-alive sessions.

Total Sessions Created in `perfdump` shows both the number of sessions that have been created and the maximum threads. Web Server can also service certain type of requests such as cached static content from the keep alive threads. Therefore, the maximum threads is the sum of the maximum threads configured in the thread-pool element in the `server.xml` file and the number of keep alive threads. For more information, see [“Accelerator Hit Ratio” on page 60](#).

The equivalent information as the Total Number of Threads is available through the Admin Console from the Monitoring tab ⇒ Instances sub tab, under General Statistics. To see the maximum threads allowed, see the Maximum Threads field on the configuration's Performance tab ⇒ HTTP sub tab, under Thread Pool Settings.

To get the equivalent of the `perfdump Active Sessions`, you can subtract the Number of Idle Threads from the Total Number of Threads.

Maximum Threads (Maximum Simultaneous Requests)

The maximum threads setting specifies the maximum number of simultaneous transactions that the Web Server can handle. The default value is greater of 128 or the number of processors in the system. Changes to this value can be used to throttle the server, minimizing latencies for the transactions that are performed. The Maximum Threads value acts across multiple virtual servers, but does not attempt to load balance. It is set for each configuration.

Reaching the maximum number of configured threads is not necessarily undesirable, and you do not need to automatically increase the number of threads in the server. Reaching this limit means that the server needed this many threads at peak load, but as long as it was able to serve requests in a timely manner, the server is adequately tuned. However, at this point connections queue up in the connection queue, potentially overloading it. If you monitor your server's performance regularly and notice that total sessions created number is often near the maximum number of threads, consider increasing your thread limits.

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 can set the maximum threads 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 generally have a very short duration. Response time can be as low as 5 milliseconds. Processing one request at a time still allows 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 IO Timeout setting on the configuration's Performance tab \Rightarrow HTTP Settings page. You can also use the command `wadm set-http-prop` and set the `io-timeout` property. The default value is 30 seconds. By setting the default value to less than 30 seconds you can free up threads sooner, but you might also disconnect users with slower connections. Also, some sites perform 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 might need to increase the number of maximum simultaneous requests.

Suitable maximum threads values range from 100-500, depending on the load. Maximum Threads represents a hard limit for the maximum number of active threads that can run simultaneously, which can become a bottleneck for performance.

The thread pool minimum threads is the minimum number of threads the server initiates upon startup. The default is set to number of processors.

Note – When configuring Web Server to be used with the Solaris Network Cache and Accelerator (SNCA), setting the maximum threads and the queue size to 0 provides better performance. Because SNCA manages the client connections, it is not necessary to set these parameters. These parameters can also be set to 0 with non-SNCA configurations, especially for cases in which short latency responses with no keep-alives must be delivered. It is important to note that the maximum threads and queue size must *both* be set to 0.

For information about using SNCA, see [“Using the Solaris Network Cache and Accelerator \(SNCA\)” on page 103](#).

Tuning

You can increase your thread limits in the Admin Console by editing the Maximum Threads field on the configuration's Performance tab ⇒ HTTP tab, under Thread Pool Settings. In the command-line interface, use the `wadm set -thread-pool-prop` command's `max-threads` property. The default value is greater of 128 or the number of processors in the system.

File Cache Statistics 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. 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. For servlets and JSPs, other kinds of caching are used.

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 slows down, the server operates normally when the cache is off.

For performance reasons, Web Server caches as follows:

- For small files, it caches the content in memory (heap).
- For medium files, it caches the content using mmap.
- For large files, it caches the open file descriptors to avoid opening and closing files.

The following is an example of how the cache statistics are displayed in `perf dump`:

```
CacheInfo:
-----
File Cache Enabled      yes
File Cache Entries      141/1024
```

File Cache Hit Ratio	652/664 (98.19%)
Maximum Age	30
Accelerator Entries	120/1024
Acceleratable Requests	281/328 (85.67%)
Acceleratable Responses	131/144 (90.97%)
Accelerator Hit Ratio	247/281 (87.90%)

The following table shows the file cache statistics as displayed in the Admin Console:

TABLE 2-4 File Cache Statistics

Total Cache Hits	46
Total Cache Misses	52
Total Cache Content Hits	0
Number of File Lookup Failures	9
Number of File Information Lookups	37
Number of File Information Lookup Failures	50
Number of Entries	12
Maximum Cache Size	1024
Number of Open File Entries	0
Number of Maximum Open Files Allowed	1024
Heap Size	36064
Maximum Heap Cache Size	10735636
Size of Memory Mapped File Content	0
Maximum Memory Mapped File Size	0
Maximum Age of Entries	30

Accelerator Entries

The number of files that have been cached in the accelerator cache.

Tuning

You can increase the maximum number of accelerator cache entries by increasing the number of file cache entries as described in [“File Cache Entries” on page 61](#). Note that this number will typically be smaller than the File Cache Entries number because the accelerator cache only caches information about files and not directories. If the number is significantly lower than the

File Cache Entries number, you can improve the accelerator cache utilization by following the tuning information described in [“Acceleratable Requests” on page 60](#) and [“Acceleratable Responses” on page 60](#).

Acceleratable Requests

The number of client requests that were eligible for processing by the accelerator cache. Only simple GET requests are processed by the accelerator cache. The accelerator cache does not process requests that explicitly disable caching, for example, requests sent when a user clicks Reload in the browser and requests that include a query string, that is, requests for URLs that include a ? character.

Tuning

To maximize the number of acceleratable requests, structure your web sites to use static files when possible and avoid using query strings in requests for static files.

Acceleratable Responses

The number of times the response to an acceleratable request was eligible for addition to the accelerator cache.

Tuning

When the server serves a static file from the file cache, the accelerator cache may be able to cache the response for faster processing on subsequent requests. To maximize performance, maximize the number of responses that can be accelerated. In the default configuration, all responses to requests for static files can be cached in the accelerator cache. The following configuration changes may prevent a response from being accelerated:

- ACLs that deny read access
- Additional directives in the default object of the `obj.conf` file, including third party plug-ins
- Using `<Client>` or `<If>` in the default object of the `obj.conf` file
- Custom access log formats
- Java Servlet filters

To maximize the number of responses that can be accelerated, avoid such configurations.

Accelerator Hit Ratio

The number of times the response for a request that can be accelerated was found in the accelerator cache. Web Server can also serve requests from the accelerator cache asynchronously directly from the keep-alive threads thereby bypassing the connection queue altogether. This leads to improved performance for these requests and at the same time reduces contention on the connection queue.

Tuning

Higher hit ratios result in better performance. To maximize the hit ratio, see the tuning information for [“Acceleratable Responses”](#) on page 60.

File Cache Enabled

If the cache is disabled, the rest of this section is not displayed in `perfdump`. In the Admin Console, the File Cache Statistics section shows zeros for the values.

Tuning

The cache is enabled by default. You can disable it in the Admin Console by deselecting the File Cache Enabled box on the configuration's Performance tab \Rightarrow Cache sub tab, under File Cache. To disable it using the command-line-interface, use `wadm set - file - cache - prop` and set the `enabled` property to `false`.

File Cache Entries

The number of current cache entries and the maximum number of cache entries are both displayed in `perfdump`. In the Admin Console, they are called the Number of Entries and the Maximum Cache Size. A single cache entry represents a single URI.

Tuning

The optimal file cache size is dependent on whether you are running a 32-bit or a 64-bit server. The available address space for a 32-bit server is limited to 4GB. The `max-entries` for file cache is based on the amount of Java heap, the number of threads (as specified by `thread-pool/max-threads`), and the connection queue size. It is recommended to cache small, frequently accessed static files in the file cache and use `perfdump` to ensure that the file cache hit ratio is close to 100%. To achieve this, you may increase file cache size and fine tune the `max-entries` for optimal performance. In a 64-bit server, due to the increased availability of address space, you can cache more static content in the file cache.

You can set the maximum number of cached entries in the Admin Console in the Maximum Entries field on the configuration's Performance tab \Rightarrow Cache tab, under File Cache. In the command-line interface, use `wadm set - file - cache - prop` and set the `max-entries` property. The default is 1024. The range of values is 1-1048576.

File Cache Hit Ratio (Cache Hits / Cache Lookups)

The hit ratio available through `perfdump` gives you the number of file cache hits compared to cache lookups. Numbers approaching 100% indicate that the file cache is operating effectively, while numbers approaching 0% indicate that the file cache is not serving many requests.

To figure this number yourself using the statistics provided through the Admin Console, divide the Total Cache Hits by the sum of the Total Cache Hits and the Total Cache Misses.

This setting is not tunable.

Maximum Age

This field displays the maximum age of a valid cache entry. The 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

Set the maximum age based on whether the content is updated (existing files are modified) on a regular schedule. For example, if content is updated four times a day at regular intervals, you can 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. If your web site's content changes infrequently, you might want to increase this value for improved performance.

Set the maximum age in the Admin Console in the Maximum Age field on the configuration's Performance tab ⇒ Cache tab, under File Cache. In the command-line interface, use `wadm set - file - cache - prop` and change the `max - age` property. The default value is 30 seconds. The range of values is 0.001-3600.

Maximum Heap Cache Size

The optimal cache heap size depends upon how much system memory is free. A larger heap size means that the Web Server can cache more content and therefore obtain a better hit ratio. However, the heap size should not be so large that the operating system starts paging cached files.

Tuning

Set the maximum heap size in the Admin Console in the Maximum Heap Space Size field on the configuration's Performance tab ⇒ Cache tab, under File Cache. In the command-line interface, use `wadm set - file - cache - prop` and change the `max - heap - space` property. The default value is 10485760 bytes. The range of values is 0-9223372036854775807. In a 32-bit Web Server, since processes have four GBs of address space for the file cache, the value should be well under four GB.

Using the nocache Parameter

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

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 previous example, the server does not cache static files from `/export/mydir/` when requested by the URL prefix `/myurl`. For more information on editing `obj.conf`, see [Sun Java System Web Server 7.0 Update 8 Administrator's Configuration File Reference](#).

Using sendfile-size to Pace Large Files

When `file-cache/sendfile` is set to true, Web Server uses the operating system `sendfile` call to service large static files from the file cache. The performance of serving such files from the file cache can be improved by limiting the number of bytes sent by a single `sendfile` call. Using a non zero value for `file-cache/sendfile-size` will send the file in chunks, where the chunk size is at the most the size of `sendfile-size`.

Tuning

While serving large files, for example, movie files, consider tuning the `file-cache/sendfile-size`. A `sendfile-size` of 1MB can provide improved performance compared to the default `sendfile-size` of zero. You can set the `sendfile-size` in Admin Console by editing Sendfile Size field on the configuration's Performance tab ⇒ Cache tab, under File Cache. In the command-line interface, use `wadm set -file-cache-prop` command's `sendfile-size` property.

File Cache Dynamic Control and Monitoring

File Cache stores file contents in the memory. You can add an object to `obj.conf` to dynamically monitor and control the file cache while the server is running.

▼ To Control and Monitor the File Cache

1 Add a NameTrans directive to the default object:

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

2 Add an nsfc object definition:

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

This configuration enables the file cache control and monitoring function (nsfc-dump) to be accessed through the URI /nfsc. To use a different URI, change the from parameter in the NameTrans directive.

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

Sun Java System File Cache Status (pid 3602)

The file cache is enabled.
Cache resource utilization

Number of cached file entries = 174968 (152 bytes each, 26595136 total bytes)
Heap space used for cache = 1882632616/1882632760 bytes
Mapped memory used for medium file contents = 0/1 bytes
Number of cache lookup hits = 47615653/48089040 (99.02 %)
Number of hits/misses on cached file info = 23720344/324195
Number of hits/misses on cached file content = 16247503/174985
Number of outdated cache entries deleted = 0
Number of cache entry replacements = 0
Total number of cache entries deleted = 0

Parameter settings

ReplaceFiles: false
ReplaceInterval: 1 milliseconds
HitOrder: false
CacheFileContent: true
TransmitFile: false
MaxAge: 3600 seconds
MaxFiles: 600000 files
SmallFileSizeLimit: 500000 bytes
MediumFileSizeLimit: 1000001 bytes
BufferSize: 8192 bytes

CopyFiles: false
Directory for temporary files: /tmp
Hash table size: 1200007 buckets

You can include a query string when you access the 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 including size and modification date 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 and appears on `shtml` files.
- `T`: Cache entry has a temporary file.
- `W`: Cache entry is locked for write access.

Thread Pool Information

If you are using the default settings, threads from the default thread pool process the request. However, you can also create custom thread pools and use them to run custom NSAPI functions. By default, Web Server creates one additional pool, named `NativePool`. In most cases, the native thread pool is only needed on the Windows platform. For more information on thread pools, see [“Understanding Threads, Processes, and Connections” on page 38](#).

Native Thread Pool

The following example shows native thread pool information as it appears in `perfdump`:

```
Native pools:
-----
NativePool:
Idle/Peak/Limit           1/1/128
Work Queue Length/Peak/Limit 0/0/0
my-custom-pool:
Idle/Peak/Limit           1/1/128
Work Queue Length/Peak/Limit 0/0/0
```

If you have defined additional custom thread pools, they are shown under the Native Pools heading in `perfdump`.

The following table shows the thread pool statistics as they appear in the Admin Console. If you have not defined additional thread pools, only the `NativePool` is shown:

TABLE 2-5 Thread Pools Statistics

Name	NativePool
Idle Threads	1
Threads	1
Requests Queued	0
Peak Requests Queued	0

Idle /Peak /Limit

Idle, listed as Idle Threads in the Admin Console, indicates the number of threads that are currently idle. Peak indicates the peak number of threads in the pool. Limit, listed as Threads in the Admin Console, indicates the maximum number of native threads allowed in the thread pool, and for NativePool is determined by the setting of NativePoolMaxThreads in the magnus.conf file.

Tuning

You can modify the maximum threads for NativePool by editing the NativePoolMaxThreads parameter in magnus.conf. For more information, see “NativePoolMaxThreads Directive” on page 67.

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, which is represented as Requests Queued in the Admin Console.

Peak indicates peak requests queued in the Admin Console and 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 queue size for NativePool by editing the NativePoolQueueSize directive in magnus.conf. For more information, see “NativePoolQueueSize Directive” on page 67.

NativePoolStackSize Directive

The NativePoolStackSize determines the stack size in bytes of each thread in the native (kernel) thread pool.

Tuning

You can modify the `NativePoolStackSize` by editing the `NativePoolStackSize` directive in `magnus.conf`.

NativePoolQueueSize Directive

The `NativePoolQueueSize` determines the number of threads that can wait in the queue for the thread pool. If all threads in the pool are busy, then the next request-handling thread that needs to use a thread in the native pool must wait in the queue. If the queue is full, the next request-handling thread that tries to get in the queue is rejected, with the result that it returns a busy response to the client. It is then free to handle another incoming request instead of being tied up waiting in the queue.

Setting the `NativePoolQueueSize` lower than the maximum threads value causes the server to execute a busy function instead of the intended NSAPI function whenever the number of requests waiting for service by pool threads exceeds this value. The default returns a “503 Service Unavailable” response and logs a message, depending on your log level setting. Setting the `NativePoolQueueSize` higher than the maximum threads causes the server to reject connections before a busy function can execute.

This value represents the maximum number of concurrent requests for service that require a native thread. If your system is unable to fulfill requests due to load, allowing more requests queue up increases the latency for requests, and could result in all available request threads waiting for a native thread. In general, set this value to be high enough to avoid rejecting requests by anticipating the maximum number of concurrent users who would execute requests requiring a native thread.

The difference between this value and the maximum threads is the number of requests reserved for non-native thread requests, such as static HTML and image files. Keeping a reserve and rejecting requests ensures that your server continues to fill requests for static files, which prevents it from becoming unresponsive during periods of very heavy dynamic content load. If your server consistently rejects connections, this value is either set too low, or your server hardware is overloaded.

Tuning

You can modify the `NativePoolQueueSize` by editing the `NativePoolQueueSize` directive in `magnus.conf`.

NativePoolMaxThreads Directive

`NativePoolMaxThreads` determine the maximum number of threads in the native (kernel) thread pool.

A higher value allows more requests to execute concurrently, but has more overhead due to context switching, so bigger is not always better. Typically, you do not need to increase this number, but if the CPU is not saturated and you see requests queue up, then increase this number.

Tuning

You can modify the `NativePoolMaxThreads` by editing the `NativePoolMaxThreads` parameter in `magnus.conf`.

NativePoolMinThreads Directive

Determines the minimum number of threads in the native (kernel) thread pool.

Tuning

You can modify the `NativePoolMinThreads` by editing the `NativePoolMinThreads` parameter in `magnus.conf`.

DNS Cache Information

The DNS cache caches IP addresses and DNS names. Web Server uses DNS caching for logging and for access control by IP address. DNS cache is enabled by default. The following example shows DNS cache information as displayed in `perfdump`:

```
DNSCacheInfo:
-----
enabled          yes
CacheEntries     4/1024
HitRatio         62854802/62862912 ( 99.99%)

AsyncDNS Data:
-----
enabled          yes
NameLookups      0
AddrLookups      0
LookupsInProgress 0
```

The following example shows the DNS Cache information as displayed in the Admin Console:

TABLE 2-6 DNS Cache Statistics

Total Cache Hits	62854802
Total Cache Misses	6110

TABLE 2-6 DNS Cache Statistics *(Continued)*

Number of Asynchronous Lookups	0
Lookups in Progress	4
Asynchronous Lookups Enabled	1
Number of Asynchronous Address Lookups Performed	0

Enabled

If the DNS cache is disabled, the rest of this section is not displayed in `perfdump`. In the Admin Console, the page displays zeros.

Tuning

By default, the DNS cache is on. You can enable or disable DNS caching in the Admin Console on the configuration's Performance tab \Rightarrow DNS sub tab, under DNS Cache Settings and selecting or deselecting the DNS Cache Enabled box. To enable or disable it using the command-line-interface, use `wadm set -dns -cache -prop` and set the `enabled` property.

Cache Entries (Current Cache Entries / Maximum Cache Entries)

This section in `perfdump` shows the number of current cache entries and the maximum number of cache entries. In the Admin Console the current cache entries are shown as Total Cache Hits. 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 access your web site concurrently. Note that setting the cache size too high wastes memory and degrades performance.

Tuning

You can set the maximum size of the DNS cache in the Admin Console in the Maximum Cache Size field on the configuration's Performance tab \Rightarrow DNS sub tab, under DNS Cache Settings. To set it using the command-line-interface, use `wadm set -dns -cache -prop` and set the `max-entries` property. The default cache size is 1024. The value range is 2-32768.

Hit Ratio of Cache Hits and Lookups

The hit ratio in `perfdump` displays the number of cache hits compared to the number of cache lookups. You can compute this number using the statistics in the Admin Console by dividing the Total Cache Hits by the sum of the Total Cache Hits and the Total Cache Misses.

This setting is not tunable.

Async DNS Enabled/Disabled

Async DNS enabled/disabled displays whether the server uses its own asynchronous DNS resolver instead of the operating system's synchronous resolver. By default, Async DNS is disabled. If it is disabled, this section does not appear in `perfdump`. To enable it using the Admin Console, on the configuration's Performance tab ⇒ DNS tab, under DNS Lookup Settings, select Asynchronous DNS. To enable it using the command-line interface, use `wadm set -dns -prop` and set the `async` property to `true`.

Java Virtual Machine (JVM) Information

JVM statistics are displayed through the Admin Console, the CLI, and `stats -xml` only. They are not shown in `perfdump`.

The following table shows an example of the JVM statistics displayed in the Admin Console:

TABLE 2-7 Java Virtual Machine (JVM) Statistics

Virtual Machine Name	Java HotSpot™ Server VM
Virtual Machine Vendor	Sun Microsystems Inc.
Virtual Machine Version	1.5.0_06-b05
Heap Memory Size	5884856
Elapsed Garbage Collection Time (milli seconds)	51
Present Number of Classes Loaded	1795
Total Number of Classes Loaded	1795
Total Number of Classes Unloaded	0
Number of Garbage Collections Occurred	3
Number of Live Threads	8
Number of Started Threads	9
Peak Live Thread Count	8

Most of these statistics are not tunable. They provide information about the JVM's operation.

Another source of tuning information on the JVM is the package `java.lang.management`, which provides the management interface for monitoring and management of the JVM. For more information on this package, see <http://java.sun.com/j2se/1.5.0/docs/api/java/lang/management/package-summary.html>.

Java Heap Tuning

As with all Java programs, the performance of the web applications in the Web Server is dependent on the heap management performed by the JVM. Balancing times and throughput is a challenge. For more information, read the performance documentation for the Java HotSpot virtual machine, which can be found at <http://java.sun.com/docs/hotspot/index.html>.

Specific documents of interest include “Tuning Garbage Collection with the 5.0 Java Virtual Machine” (http://java.sun.com/docs/hotspot/gc5.0/gc_tuning_5.html) and “Ergonomics in the 5.0 Java Virtual Machine” (<http://java.sun.com/docs/hotspot/gc5.0/ergo5.html>).

JVM options can be specified in the Admin Console on the configuration's Java tab ⇒ JVM Settings sub tab. In the CLI, use the wadm commands `set-jvm-prop` and `set-jvm-profiler-prop`.

Web Application Information

Web application statistics are displayed through the Admin Console, `wadm get-config-stats` command), and `stats-xml` only. They are not shown in `perfdump`.

▼ To Access Web Application Statistics From the Admin Console

- 1 From the Common Tasks page, choose the Monitoring tab.
- 2 Click the configuration name to view web application statistics for the configuration. To view web application statistics for the instance, click the Instance sub tab and the instance name.
- 3 On the Monitoring Statistics page, click Virtual Server Statistics.
- 4 Click the virtual server name.
- 5 On the Virtual Server Monitoring Statistics page, click Web Applications.
- 6 Select the web application for which to view statistics from the Web Application pull-down menu.

Web Application Statistics

The following table shows an example of the Web Application statistics displayed in the Admin Console:

TABLE 2-8 Web Application Statistics

Number of JSPs Loaded	1
Number of JSPs Reloaded	1
Total Number of Sessions Serviced	2
Number of Sessions Active	2
Peak Number of Active Sessions	2
Number of Sessions Rejected	0
Number of Sessions Expired	0
Average Time (seconds) that expired sessions had been alive	0
Longest Time (seconds) for which an expired session was alive	0

For more information on tuning, see [“Tuning Java Web Application Performance” on page 78](#). Also see [Sun Java System Web Server 7.0 Update 8 Developer’s Guide to Java Web Applications](#).

JDBC Resource Information

A JDBC resource is a named group of JDBC connections to a database. A JDBC resource defines the properties used to create a connection pool. Each JDBC resource uses a JDBC driver to establish a connection to a physical database when the server is started. A pool of connections is created when the first request for connection is made on the pool after you start Web Server.

A JDBC-based application or resource draws a connection from the pool, uses it, and when no longer needed, returns it to the connection pool by closing the connection. If two or more JDBC resources point to the same pool definition, they use the same pool of connections at run time.

The use of connection pooling improves application performance by doing the following:

- Creating connections in advance. The cost of establishing connections is moved outside of the code that is critical for performance.
- Reusing connections. The number of times connections are created is significantly lowered.
- Controlling the amount of resources a single application can use at any moment.

JDBC resources can be created and edited using the Admin Console's Java tab ⇒ Resources sub tab for the configuration. You can also use the `wadm create-jdbc-resource` and `set-jdbc-resource-prop` commands. For more information, see the [Sun Java System Web Server 7.0 Update 8 Administrator’s Guide](#).

Note – Each defined pool is instantiated during Web Server startup. However, the connections are only created the first time the pool is accessed. Jump-start a pool before putting it under heavy load.

JDBC resource statistics are available through the Admin Console, CLI, and `stats.xml` only. They are not shown in `perfdump`. Some of the monitoring data is unavailable through the Admin Console and can only be viewed through the CLI using `wadm get-config-stats` and through the `stats.xml` output.

A pool is created on demand, that is, it is created the first time it is used. The monitoring statistics are not displayed until the first time the pool is used.

JDBC Resource Statistics Available Through the Admin Console

The following table shows an example of the JDBC resource statistics displayed through the Admin Console:

TABLE 2-9 JDBC Resource Statistics

Connections	32
Free Connections	0
Leased Connections	32
Average Queue Time	1480.00
Queued Connections	40
Connection Timeout	100

To change the settings for a JDBC resource through the Admin Console, for the configuration, choose the Java tab ⇒ Resources sub tab. Select the JDBC resource. The settings are available on the Edit JDBC Resource page. To change the JDBC resource through the command-line-interface, use `wadm set-jdbc-resource-prop`.

Connections

This number shows the current JDBC connections, including both free and busy connections.

Tuning – This setting cannot be tuned, but it is a good indicator of recent pool activity. If the number of connections is consistently higher than the minimum number of connections, consider increasing the minimum number of connections to be closer to the number of current JDBC connections. To change the minimum connections for a JDBC resource through the Admin Console, on the Edit JDBC Resources page, edit the Minimum Connections setting. To change the JDBC resource's minimum connections through the command-line-interface, use `wadm set-jdbc-resource-prop` and change the `min-connections` property.

Free Connections

This number shows the current number of free connections in the pool. All free connections over the minimum pool size are closed if they are idle for more than the maximum idle timeout. The free connections are not tunable.

Leased Connections

This number shows the current number of connections in use.

Tuning – If number of leased connections is consistently lower than the minimum connections, consider reducing the minimum connections for the JDBC resource. If number of leased connections is consistently higher than minimum connections, consider increasing the minimum connections. If number of leased connections is consistently at the JDBC resource's maximum number of connections, consider increasing the maximum number of connections. The upper limit for the number of leased connections is the number of maximum connections.

To change the minimum or maximum connections for a JDBC resource through the Admin Console, on the Edit JDBC Resource page, edit the Minimum Connections or Maximum Connections fields. To change the JDBC resource's minimum or maximum connections through the command-line-interface, use `wadm set - jdbc-resource-prop` and change the `min-connections` or `max-connections` properties.

Queued Connections

This number shows the current number of requests for connections that are waiting to receive a connection from the JDBC pool. Connection requests are queued if the current number of leased connections has reached the maximum connections.

Tuning – If this number is consistently greater than zero, consider increasing the JDBC resource's maximum connections. To change the maximum connections for a JDBC resource through the Admin Console, edit the Maximum Connections field on the Edit JDBC Resource page. To change the JDBC resource's maximum connections through the command-line-interface, use `wadm set - jdbc-resource-prop` and change the `max-connections` property.

JDBC Resource Statistics Not Available in the Admin Console

Some JDBC statistics are available through the `wadm get - config-stats` command using the `- -node` option, through `stats -xml`, and through SNMP, but not through the Admin Console.

maxConnections – The configured maximum size of the pool. Use as a reference for other statistics. To change the maximum connections for a JDBC resource through the Admin Console, on the Edit JDBC Resource page, edit the Maximum Connections field. To change the JDBC resource's maximum connections through the command-line-interface, use `wadm set - jdbc-resource-prop` and change the `max-connections` property.

peakConnections – The highest number of connections that have been leased concurrently during the history of the pool. This number is a good indication on the upper limit on pool usage. It is limited by the maximum connections setting.

countTotalLeasedConnections – The total number of times a connection has been handed out by the pool. Indicates total pool activity. Not tunable.

countTotalFailedValidationConnections – If connection validation is enabled, shows the number of times a connection has been detected as invalid by the pool. If this number is relatively high, it could signal database or network problems. Not tunable.

peakQueued – The highest number of connection requests that have been queued simultaneously at any time during the lifetime of the pool. Not tunable.

millisecondsPeakWait – The maximum time in milliseconds that any connection request has been in the wait queue. A high number is an indication of high pool activity. The upper limit is the JDBC resource setting wait timeout.

countConnectionIdleTimeouts – The number of free connections that have been closed by the pool because they exceeded the configured JDBC idle timeout. To change the idle timeout for a JDBC resource through the Admin Console, on the Edit JDBC Resource page, edit the Idle Timeout field. To change the JDBC resource's idle timeout through the command-line-interface, use `wadm set -jdbc-resource-prop` and change the `idle-timeout` property.

JDBC Resource Connection Settings

Depending on your application's database activity, you might need to size JDBC resource connection pool settings. Attributes of a JDBC resource which affect performance are listed below, along with performance considerations when setting values.

- **Minimum connections**

The size the pool keeps during the life of the server instance, and the initial size of the pool. Defaults to 8. This number should be as close as possible to the expected average size of the pool. Use a high number for a pool that is expected to be under heavy load, to minimize creation of connections during the life of the application and minimize pool resizing. Use a lower number if the pool load is expected to be small, to minimize resource consumption.

- **Maximum connections**

The maximum number of connections that a pool can have at any given time. Defaults to 32. Use this setting to enforce a limit in the amount of connection resources that a pool or application can have. This limit is also beneficial to avoid application failures due to excessive resource consumption.

- **Idle timeout**

The maximum amount in seconds that a connection is ensured to remain unused in the pool. After the idle timeout, connections are automatically closed. If necessary, new connections are created up to the minimum number of connections to replace the closed connection. Note that this setting does not control connection timeouts enforced at the database server side. Defaults to 60 seconds.

Setting this attribute to `-1` prevents the connections from being closed. This setting is used for pools that expect continuous high demand. Otherwise, keep this timeout shorter than the database server-side timeout (if such timeouts are configured on the specific vendor database), to prevent accumulation of unusable connections in the pool.

- **Wait timeout**

The amount of time in seconds that a request waits for a connection in the queue before timing out. After this timeout, the user sees an error. Defaults to 60.

Setting this attribute to `-1` causes a request for a connection to wait indefinitely. This setting could also improve performance by keeping the pool from having to account for connection timers.

- **Validation method**

The method used by the pool to determine the health of a connections in the pool. Defaults to `off`.

If a validation method is used, the pool executes a sanity check on a connection before leasing it to an application.

The effectiveness and performance impact depends on the method selected:

- `meta-data` is less expensive than `table` in terms of performance, but usually less effective as most drivers cache the result and do not use the connection, providing false results.
- `table` is almost always effective, as it forces the driver to perform an SQL call to the database, but it is also the most costly.
- `auto-commit` can provide the best balance of effectiveness and performance cost, but a number of drivers also cache the results of this method.

- **Validation Table Name**

The user-defined table to use for validation when the validation method is `table`. Defaults to `test`.

If this method is used, the table used should be dedicated only to validation, and the number of rows in the table should be kept to a minimum.

- **Fail All Connections**

Indicates whether all connections in the pool are re-created when one is found to be invalid, or only the invalid one. Only applicable if you have selected a connection validation method. Disabled by default.

If enabled, all of the re-creation is done in one step, and the thread requesting the connection is heavily affected. If disabled, the load of re-creating connections is distributed between the threads requesting each connection.

- Transaction Isolation Level

Specifies the Transaction Isolation Level on the pooled database connections.

By default, the default isolation level of the connection is left intact. Setting it to any value does incur the small performance penalty caused by the method call.

- Guarantee Isolation

Only applicable if a transaction isolation level is specified. Defaults to disabled.

Leaving this setting disabled causes the isolation level to be set only when the connection is created. Enabling sets the level every time the connection is leased to an application. In most cases, leave this setting disabled.

Tuning the ACL User Cache

The ACL user cache is active 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, Web Server must query the LDAP server more often to validate users, which impacts performance.

This bottleneck can be avoided by increasing the maximum users of the ACL cache on the configuration's Performance tab ⇒ Cache sub tab. You can also set the number of users by setting the `max-users` property using the command `wadm set -acl -cache -prop`. Note that increasing the cache size uses 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 (four by default). 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 Maximum Groups setting on the configuration's Performance tab ⇒ Cache sub tab. Or you can use the `max-groups-per-user` property of the `wadm set -acl -cache -prop` command.

The maximum age setting of the ACL cache 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 the maximum age setting. The entry is not used if its age is greater than or equal to the maximum age. The default value is 120 seconds. If your LDAP is not likely to change often, use a

large number for the maximum age. However, if your LDAP entries change often, use a smaller value. For example, when the value is 120 seconds, the Web Server might be out of sync with the LDAP server for as long as two minutes. Depending on your environment, that might or might not be a problem.

Tuning Java Web Application Performance

This section contains information to help you improve the performance of your Java Web Applications. This section includes the following topics:

- [“Using Precompiled JSPs” on page 78](#)
- [“Using Servlet/JSP Caching” on page 79](#)
- [“Configuring the Java Security Manager” on page 79](#)
- [“Configuring Class Reloading” on page 79](#)
- [“Avoiding Directories in the Classpath” on page 79](#)
- [“Configuring the Web Application’s Session Settings” on page 80](#)

In addition, see the following sections for other tuning information related to the Java Web Applications:

- [“Java Virtual Machine \(JVM\) Information” on page 70](#)
- [“JDBC Resource Information” on page 72](#)

Using Precompiled JSPs

Compiling JSPs is a resource-intensive and relatively time-consuming process. By default, the Web Server periodically checks to see if your JSPs have been modified and dynamically reloads them. This allows you to deploy modifications without restarting the server. The `reload-interval` property of the `jsp-config` element in `sun-web.xml` controls how often the server checks JSPs for modifications. However, there is a small performance penalty for that checking.

When the server detects a change in a `.jsp` file, only that JSP is recompiled and reloaded; the entire web application is not reloaded.

If your JSPs don't change, you can improve performance by pre-compiling your JSPs.

When adding a web application, either through the Admin Console or CLI, choose the precompile JSPs option. Enabling precompiled JSPs allows all the JSPs present in the web application to be pre-compiled, and their corresponding servlet classes are grouped in the web application's `WEB-INF/lib` or `WEB-INF/classes` directory. When a JSP is accessed, it is not compiled and instead, its pre-compiled servlet is used. For more information on JSPs, see [Sun Java System Web Server 7.0 Update 8 Developer's Guide to Java Web Applications](#). Also see [“Configuring Class Reloading” on page 79](#).

Using Servlet/JSP Caching

If you spend a lot of time re-running the same servlet/JSP, you can cache its results and return results out of the cache the next time it is run. This is useful for common queries that all visitors to your site run: you want the results of the query to be dynamic because the results might change daily, but you don't need to run the logic for every user.

To enable caching, you configure the caching parameters in the `sun-web.xml` file of your application. For more details, see [“Caching Servlet Results” in *Sun Java System Web Server 7.0 Update 8 Developer’s Guide to Java Web Applications*](#).

Configuring the Java Security Manager

Web Server supports the Java Security Manager. The main drawback of running the Security Manager is that it negatively impacts performance. The Java Security Manager is disabled by default when you install the product. Running the Web Server without the Security Manager might improve performance significantly for some types of applications. Based on your application and deployment needs, you should evaluate whether to run the Web Server with or without the Security Manager. For more information, see [Sun Java System Web Server 7.0 Update 8 Developer’s Guide to Java Web Applications](#).

Configuring Class Reloading

The dynamic reload interval of the servlet container and the `dynamic-reload-interval` of the `class-loader` element in `sun-web.xml` control the frequency at which the server checks for changes in servlet classes. When dynamic reloading is enabled and the server detects that a `.class` file has changed, the entire web application is reloaded.

Set the dynamic reload interval on the configuration's Java tab ⇒ Servlet Container sub tab, or using the `wadm set-servlet-container-props` command. In a production environment where changes are made in a scheduled manner, set this value to 0 to prevent the server from constantly checking for updates. The default value is 0 (that is, class reloading is disabled). For more information about elements in `sun-web.xml`, see [Sun Java System Web Server 7.0 Update 8 Developer’s Guide to Java Web Applications](#).

Avoiding Directories in the Classpath

For certain applications and especially if the Java Security Manager is enabled you can improve performance by ensuring that there are no unneeded directories in the classpath. To do so, change the Server Class Path, Class Path Prefix, and Class Path Suffix fields on the configuration's Java tab ⇒ General sub tab for the configuration or use the command `wadm set-jvm-prop`. Also, package the web application's `.class` files in a `.jar` archive in

WEB-INF/lib instead of packaging the .class files as is in WEB-INF/classes, and ensure that the .war archive does not contain a WEB-INF/classes directory.

Configuring the Web Application's Session Settings

If you have relatively short sessions, try decreasing the session timeout by configuring the value of the `timeOutSeconds` property under the `session-properties` element in `sun-web.xml` from the default value of 10 minutes.

If you have relatively long sessions, you can try decreasing the frequency at which the session reaper runs by increasing the value of the `reapIntervalSeconds` property from the default value of once every minute.

For more information about these settings, and about session managers, see [Sun Java System Web Server 7.0 Update 8 Developer's Guide to Java Web Applications](#).

In multi-process mode when the `persistence-type` in `sun-web.xml` is configured to `mmap`, the session manager uses cross-process locks to ensure session data integrity. These can be configured to improve performance as described below.

Note – For Java technology-enabled servers, multi-process mode is deprecated and included for backward-compatibility only.

Tuning `maxLocks` (UNIX/Linux)

The implication of the number specified in the `maxLocks` property can be gauged by dividing the value of `maxSessions` with `maxLocks`. For example, if `maxSessions = 1000` and you set `maxLocks = 10`, then approximately 100 sessions (1000/10) contend for the same lock. Increasing `maxLocks` reduces the number of sessions that contend for the same lock and might improve performance and reduce latency. However, increasing the number of locks also increases the number of open file descriptors, and reduces the number of available descriptors that would otherwise be assigned to incoming connection requests.

For more information about these settings, see [Chapter 6, “Session Managers,” in Sun Java System Web Server 7.0 Update 8 Developer's Guide to Java Web Applications](#).

Tuning `MMapSessionManager` (UNIX/Linux)

The following example describes the effect on process size when configuring the `persistence-type="mmap"` using the `manager-properties` properties. For more information, see [“MMap Session Manager \(UNIX Only\)” in Sun Java System Web Server 7.0 Update 8 Developer's Guide to Java Web Applications](#).

```
maxSessions = 1000
maxValuesPerSession = 10
maxValueSize = 4096
```


This example creates a memory mapped file of size 1000 X 10 X 4096 bytes, or ~40 MB. As this is a memory mapped file, the process size will increase by 40 MB upon startup. The larger the values you set for these parameters, the greater the increase in process size.

Tuning CGI Stub Processes (UNIX/Linux)

In Web Server, the CGI engine creates CGI stub processes as needed. On systems that serve a large load and rely heavily on CGI-generated content, it is possible for the CGI stub processes to consume all system resources. If this happens on your server, the CGI stub processes can be tuned to restrict how many new CGI stub processes can be spawned, their timeout value, and the minimum number of CGI stub process that run at any given moment.

Note – If you have an `init-cgi` function in the `magnus.conf` file running in multi-process mode, you must add `LateInit = yes` to the `init-cgi` line.

Tune the following settings to control CGI stubs. These settings are on the configuration's Performance Tab ⇒ CGI sub tab.

- **Minimum Stubs Size:** Controls the number of processes that are started by default. The first CGI stub process is not started until a CGI program has been accessed. The default value is 0. If you have an `init-cgi` directive in the `magnus.conf` file, the minimum number of CGI stub processes are spawned at startup.
- **Maximum Stub Size:** Controls the maximum number of CGI stub processes the server can spawn. This is the maximum concurrent CGI stub processes in execution, not the maximum number of pending requests. The default value is 16 and should be adequate for most systems. Setting this value too high might actually reduce throughput.
- **CGI Stub Timeout:** Causes the server to kill any CGI stub processes that have been idle for the number of seconds set by this directive. Once the number of processes is at the minimum stubs size, it does not kill any more processes. The default is 30.
- **CGI Timeout:** Limits the maximum time in seconds that CGI processes can run. The default is -1, which means there is no timeout.

Using find-pathinfo-forward

The `find-pathinfo-forward` parameter used in `obj.conf` can help improve performance. It is used with the `PathCheck` function `find-pathinfo` and the `NameTrans` functions `pfx2dir` and `assign-name`. The `find-pathinfo-forward` parameter instructs the server to search forward for `PATH_INFO` in the path after `nttrans-base`, instead of backward from the end of the path in the server function `find-pathinfo`. The `find-pathinfo-forward` parameter is not supported within the Admin GUI. So, users need to manually edit the corresponding `obj.conf` file and upload the changes to the admin config repository.

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.

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`. On the Windows platform, you can also use this feature to prevent the server from changing `"\"` to `"/"` when using the `PathCheck` server function `find-pathinfo`.

For more information about `obj.conf`, see the [Sun Java System Web Server 7.0 Update 8 Administrator's Configuration File Reference](#).

Using nostat

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

```
nostat=virtual-path
```

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 previous example, the server does not obtain statistics for path `/ntrans-base/nsfc` and `/ntrans-base/nsfc/*` if `ntrans-base` is set. If `ntrans-base` is not set, the server does not obtain statistics 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-name` `NameTrans`, the server assumes that the statistics 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 statistics on those URLs.

For more information about `obj.conf`, see the [Sun Java System Web Server 7.0 Update 8 Administrator's Configuration File Reference](#).

Using Busy Functions

The default busy function returns a "503 Service Unavailable" response and logs a message depending upon the log level setting. You might want to modify this behavior for your application. You can specify your own busy functions for any NSAPI function in the `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 function allows different responses if the server become too busy in the course of processing a request that includes a number of types (such as `Service`, `AddLog`, and `PathCheck`). Note that the busy function applies 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 Large Pages Support

In Sun Java System Web Server (especially in 64-bit systems), as the process address space becomes bigger and bigger, the number of Translation Lookup Buffer (TLB) misses increases significantly thereby slowing down the server.

A significant performance gain of 4–8% is observed in Sun Java System Web Server, by enabling 256M page support on CMT systems. It is observed that while running `specweb ecommerce php` on Web Server, about 7% of the CPU time is spent to handle TLB misses (`trapstat` output). Whereas, while running the same on Web Server CMT systems with 256M page support, the CPU time spent to handle TLB misses is reduced to 2%.

The Solaris 9 and 10 systems, have Multiple Page Size Support (MPSS) feature which significantly improves performance of a wide range of applications, by reducing the wastage of CPU cycles in serving TLB misses.

To enable mpss support in 64-bit Web Server on CMT, add the following in bin/startserv:

```
LD_PRELOAD_64=/usr/lib/sparcv9/mpss.so.1; export LD_PRELOAD_64
MPSSHEAP=256M; export MPSSHEAP
```

To enable mpss support in 32-bit Web Server on CMT, add the following in bin/startserv:

```
LD_PRELOAD=/usr/lib/mpss.so.1; export LD_PRELOAD
MPSSHEAP=256M; export MPSSHEAP
```

For systems which have only 4M pages, add the following in bin/startserv:

```
LD_PRELOAD_64=/usr/lib/64/mpss.so.1; export LD_PRELOAD_64
MPSSHEAP=4M; export MPSSHEAP
```

The pmap -sx command provides information on the Web Server page sizes. From the command output, note that a few page sizes are mapped with 256M pages while others are mapped with 64K pages.

```
# pmap -sx 1938

1938:  webservd -d /opt/SUNWwbsvr/https-nsapihp/config -r /opt/SUNWwbsvr -t
      Address      Kbytes      RSS      Anon      Locked Pgsz Mode   Mapped File
...
0000000100B00000      6080       6080        -        -    - rwx--   [ heap ]
00000001010F0000     244800     244800     244224        -   64K rwx--   [ heap ]
0000000110000000     3932160     3932160     3932160        -  256M rwx--   [ heap ]
0000000200000000     262144     262144     262144        -  256M rwx--   [ heap ]
FFFFFFFFF6800000      4096       4096       4096        -   64K rwx--   [ anon ]
...
```

Note – pagesize -a command is used to determine the maximum page sizes supported on your system.

Tuning Your Web Application

This section provides information on tuning applications for maximum performance. A complete guide to writing high performance Java and Java 2 EE Web applications is beyond the scope of this document.

Java Programming Guidelines

This section covers issues related to Java coding and performance. The guidelines outlined are not specific to Sun Java System Web Server, but are general rules that are useful in many situations. For a complete discussion of Java coding best practices, see the Java Blueprints.

Avoid Serialization and Deserialization

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. Additionally, customized `readObject()` and `writeObject()` methods may be beneficial in some cases.

Use StringBuffer to Concatenate Strings

To improve performance, instead of using string concatenation, use `StringBuffer.append()`. String objects are immutable; they never change after creation. For example, consider the following code:

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

The compiler translates this code as:

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

Therefore, copying is inherently expensive and overusing it can reduce performance significantly.

Assign `null` to Variables That Are No Longer Needed

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

An application can induce memory leaks by not releasing object references. Doing so prevents the Java garbage collector from reclaiming those objects, and results in increasing amounts of memory being used. Explicitly nullifying references to variables after their use 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 leak-free application in steady state will show a steady active heap memory after garbage collections.

Declare Methods as final Only If Necessary

Modern optimizing dynamic compilers can perform inlining and other inter-procedural optimizations, even if Java methods are not declared final. Use the keyword `final` as it was originally intended: for program architecture reasons and maintainability.

Only 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 you declare constants as static final variables.

Avoid Finalizers

Adding finalizers to 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.HashMap`.

Use DataHandlers for SOAP Attachments

Using a `javax.activation.DataHandler` for a SOAP attachment improves performance.

JAX-RPC specifies:

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

As a result, send a SOAP attachment as a .gif or XML file to an RPC style web service by utilizing the Java type mappings. When passing in any of the mandated Java type mappings which are appropriate for the attachment's MIME type as an argument for the web service, the JAX-RPC runtime handles these as SOAP attachments. For example, to send out an image or a gif attachment, use `java.awt.Image`, or 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:** Informal tests have shown that using `DataHandler` wrappers doubles throughput for image or GIF MIME types, and multiplies throughput by approximately 1.5 for text and XML or `java.awt.Image` for image/* types.

Java Server Page and Servlet Tuning

This section describes how to improve performance of web applications, both through coding practices and through deployment and configuration settings.

Suggested Coding Practices

This section provides some tips on coding practices that improve servlet and JSP application performance.

General Guidelines:

Follow these general guidelines to increase performance of the presentation tier:

- Minimize Java synchronization in servlets.
- Do not use the single thread model for servlets.
- Use the servlet's `init()` method to perform expensive one-time initialization.
- Avoid using `System.out.println()` calls.

Avoid Shared Modified Class Variables

In the servlet multithread model which is the default, 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.

HTTP Session Handling

Follow these guidelines when using HTTP sessions:

- Create sessions sparingly. Session creation is not free. If a session is not required, do not create one.
- Use `javax.servlet.http.HttpSession.invalidate()` to release sessions when they are no longer needed.
- Keep session size small, to reduce response times. If possible, keep session size below seven KB.
- Use the directive `<%page session="false"%>` in JSP files to prevent the Server from automatically creating sessions when they are not necessary.
- Avoid large object graphs in an `HttpSession`. They force serialization and add computational overhead. Generally, do not store large objects as `HttpSession` variables.
- Do not cache transaction data in `HttpSession`. Access to data in an `HttpSession` is not transactional. Do not use it as a cache of transactional data, which is better kept in the database.

Tuning Web Container Within Web Server 7.0

To deploy Java web applications within Web Server 7.0, you need to perform the following tuning within your server:

- [“Tuning the Garbage Collector” on page 90](#)
- [“Tuning the Java Heap” on page 91](#)
- Tuning web application configuration settings, see [“Web Server Tuning” on page 119](#) and the values in `default-web.xml` like `reload interval`.

This section describes how to improve performance of web applications, through deployment and configuration settings.

Deployment Settings

Deployment settings can have significant impact on performance. Follow these guidelines when configuring deployment settings for best performance:

- Use Pre-compiled JavaServer Pages
- Disable Dynamic Application Reloading

Use Pre-compiled JavaServer Pages

Compiling JSP files is resource intensive and time consuming. Precompiling JSP files before deploying applications on the server will improve application performance. When you compile in advance, only the resulting servlet class files will be deployed.

You can specify on the server to pre-compile JSP files when you deploy an application through the Admin Console, through CLI or GUI. You can also specify on the server to pre-compile JSP files for an already deployed application with the Admin Console.

Disable Dynamic Application Reloading

If dynamic reloading is enabled, the server periodically checks for changes in deployed applications and automatically reloads the application with the changes. Dynamic reloading is intended for development environments and is also incompatible with session persistence. To improve performance, disable dynamic class reloading.

Disable dynamic class reloading for an application

Remove `.reload` from the web application directory.

Disable Dynamic JSP Reloading

On a production system, improve web container performance by disabling dynamic JSP reloading. To do so, edit the `default-web.xml` file in the `config` directory for each instance. Change the servlet definition for a JSP file to look like this:

```
<servlet>
  <servlet-name>jsp</servlet-name>
  <servlet-class>org.apache.jasper.servlet.JspServlet</servlet-class>
  <init-param>
    <param-name>httpMethods</param-name>
    <param-value>GET, HEAD, POST</param-value>
  </init-param>
  <init-param>
    <param-name>fork</param-name>
    <param-value>>false</param-value>
  </init-param>
```

```
<init-param>
  <param-name>mappedfile</param-name>
  <param-value>>false</param-value>
</init-param>
<init-param>
  <param-name>developments</param-name>
  <param-value>>false</param-value>
</init-param>
<init-param>
  <param-name>genStrAsCharArray</param-name>
  <param-value>>true</param-value>
</init-param>
<init-param>
  <param-name>trimSpaces</param-name>
  <param-value>>true</param-value>
</init-param>
<load-on-startup>3</load-on-startup>
</servlet>
```

Note – The suggested manual changes requires you to run either `pull-config` through the CLI or the Admin Console.

Logger Settings

The Server produces writes log messages and exception stack trace output to the log file in the logs directory of the instance, Naturally, the volume of log activity can impact server performance; particularly in benchmarking situations.

Managing Memory and Garbage Collection

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

- Goals
- Tracing Garbage Collection
- Other Garbage Collector Settings
- Tuning the Java Heap
- Re-basing DLLs on Windows

Tuning the Garbage Collector

Monitoring the Garbage Collection (GC) activity at the development server and accordingly tuning JVM and GC settings before deploying the server into production is necessary. The GC settings vary depending on the application you are running.

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 and consume as much as 25 percent throughput.

Other Garbage Collector Settings

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

Tuning the Java Heap

The heap space is divided into old and the new generations. The new generation includes the new object space (eden), and two survivor spaces. The JVM allocates new objects in the eden space, and moves longer lived objects from the new generation to the old generation. Keep the heap size low, so that customers can increase the heap size depending on their needs. To increase the heap size, refer to the link, <http://www.devx.com/tips/Tip/5578>

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, meaning they are copied to the tenured generation. The tenured generation is larger and fills up less quickly. Garbage is 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.

The frequent young space collections are quick, lasting only a few milliseconds, while the full generation collection takes a longer, tens of milliseconds to a few seconds, depending upon the heap size. Other GC algorithms, such as the Concurrent Mark Sweep (CMS) algorithm, are incremental. They divide the full GC into several incremental pieces. This provides a high probability of small pauses. This process 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. So custom object pooling is not often required. Consider pooling only if object construction cost is high and significantly affects execution profiles.

The `-Xms` and `-Xmx` parameters define the minimum and maximum heap size. As collections occur when generations fill up, throughput is inversely proportional to the available memory. By default, JVM grows or shrinks the heap at each collection. This helps maintain the proportion of free space to living object at each collection within a specific range. The range is

set as a percentage by the parameters `-XX:MinHeapFreeRatio=<minimum>` and `-XX:MaxHeapFreeRatio=<maximum>`; and the total size is bound by `-Xms` and `-Xmx`.

JVM heap setting for Web Server should be based on the available memory on the system and frequency and duration of garbage collection. You can use `-verbose:gc jvm` option or the J2SE 5.0 monitoring tools to determine the frequency of garbage collection. For more information on J2SE 5.0 monitoring tools, see “[J2SE 5.0 Monitoring Tools](#)” on page 93. The maximum heap size should be determined based on the process data model (32-bit or 64-bit) and availability of virtual and physical memory on the system. Excessive use of physical memory for Java heap may cause paging of virtual memory to disk during garbage collection, resulting in poor performance. For more information on Java tuning, see <http://java.sun.com/performance/reference/whitepapers/tuning.html>.

Choosing the Garbage Collection Algorithm

This section describes how to use different garbage collector within Web Server.

CMS Collector

Use the CMS collector as the GC algorithm. This collector can cause a drop in throughput for heavily utilized systems, because it is running constantly, but it prevents the long pauses that can occur when the garbage collector runs infrequently.

Procedure to use the CMS collector:

1. Shut down the server.
2. Configure the CMS collector in the server instance.

add the following JVM options either using the Admin Console or by using the CLI:

```
-XX:+UseConcMarkSweepGC  
-XX:SoftRefLRUPolicyMSPerMB=1
```

Use the `jvmstat` utility to monitor HotSpot garbage collection.

For detailed information on tuning the garbage collector, see Tuning Garbage Collection with the 5.0 Java Virtual Machine. See (http://java.sun.com/docs/hotspot/gc5.0/gc_tuning_5.html#1.1.Sizing%20the%20Generations%7Coutline).

Tracing Garbage Collection

The 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 GC. Pauses are times when an application appears unresponsive due to GC.

Two other considerations are footprint and promptness. Footprint is the working size of the JVM 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 size makes a trade-off between these four metrics. For example, a large young generation likely maximizes throughput, but at the cost of footprint and promptness.

J2SE 5.0 Monitoring Tools

J2SE 5.0 supports tools that can be used to provide information on performance and resource consumption for running applications. Two commonly used tools, `jconsole` and `jstat`, are discussed in this section.

For a complete list of available tools and their usage, see [J2SE 5.0 Trouble-Shooting and Diagnostic Guide](#).

jconsole

The JMX based graphical tool `jconsole` is used to display thread usage, memory consumption, and class loading details. For more information about this tool, see <http://java.sun.com/j2se/1.5.0/docs/guide/management/jconsole.html>.

To use the tool, you need to:

1. Set the JVM option `-Dcom.sun.management.jmxremote` in `server.xml`.

Use the Admin Console or CLI to set this JVM option. For more information, see “[Configuring Your JVM](#)” in *Sun Java System Web Server 7.0 Update 8 Administrator’s Guide*.

2. Start `jconsole`.

The binary is available in `$JAVA_HOME/bin` directory. Specify the `pid` in the command line when invoking the tool. The `pid` of the Web Server process is obtained either from the log file or by locating the child `webserverd` process.

You may also invoke `jconsole` without the `pid` argument and then select the appropriate VM from the window displaying available VMs.

jstat

The `jstat` utility can be used to tune the Java heap size and diagnose performance issues related to heap and garbage collection. This utility can be used without any server configuration changes as the VM instrumentation is enabled by default. For more information, see <http://java.sun.com/j2se/1.5.0/docs/tooldocs/share/jstat.html>.

Binary Logging Header

Binary logging is a functionality introduced in Sun Java System Web Server 7.0 Update 5. This feature allows server information to be stored in a single log file that contains binary, unformatted log data of all the web sites hosted on a server. It thus minimizes the usage of

system resources used for logging, may improve performance and scalability, and at the same time records detailed log information. The feature is enabled by making changes in the `server.xml` file. The `server.xml` should have a new element `mode` with binary value specified in `access-log`. If the `mode` element is missing or its value is not specified, the default value is `text`. The `text` option writes access log in ASCII format, as was in previous Web Server releases. However, if the `mode` element is specified with binary value, the access log is written in binary format.

To enable binary logging, add the following in the `server.xml`:

```
<access-log>
<file>../logs/access/</file>
<mode>binary</mode>
</access-log>
```

In Sun Java System Web Server 7.0 , the binary log file is bundled in the product `bin` directory called `binlog` and includes a log writer 1.0, log reader 1.0 and a version identifier. The version identifier identifies the versions of log writer and log reader and will be upgraded accordingly in future releases.

The log writer 1.0 writes the log format into a raw binary log file which is not human readable. The binary log reader 1.0 takes the binary access log file as input, extracts it and prints the log in human readable ASCII format on the console.

Binary log accepts the following arguments:

Argument	Description
-h	Displays the help.
-v	Displays the version number of the binary log reader.
-i	It is the input binary log file name. The default value is <code>stdin</code> .

Binary log is executed in the following two ways:

```
$binlog < access.log
$binlog -i access.log
```

Common Performance Problems

This chapter discusses common web site performance problems, and includes the following topics:

- “[check-acl Server Application Functions](#)” on page 95
- “[Low-Memory Situations](#)” on page 96
- “[Too Few Threads](#)” on page 96
- “[Cache Not Utilized](#)” on page 97
- “[Keep-Alive Connections Flushed](#)” on page 97
- “[Log File Modes](#)” on page 98
- “[Tuning of File Descriptors](#)” on page 98

Note – For platform-specific issues, see [Chapter 4, “Platform-Specific Issues and Tips”](#)

check-acl **Server Application Functions**

For optimal server performance, use ACLs only when required.

The 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 performance by removing the `check-acl` SAF from the default object for URIs that are not protected by ACLs.

Low-Memory Situations

If Web Server must run in low-memory situations, reduce the thread limit to a bare minimum by lowering the value of the Maximum Threads setting on the configuration's Performance Tab ⇒ HTTP sub tab. You can also set it with `wadm set - thread - pool - prop` command's `max - threads` property.

The server automatically selects many server defaults based on the system resources, for optimal performance. However, if the server's chosen defaults are not suited to your configuration, you can override them. For more information about how to tune the server to obtain a smaller memory footprint, see [“Large Memory Footprint” on page 98](#).

Web applications running under stress might sometimes result in the server running out of Java VM runtime heap space, as seen in the `java.lang.OutOfMemoryError` messages in the server log file. There can be several reasons for this, including excessive allocation of objects, and such behavior can affect performance. To address this problem, profile the application. Refer to the following HotSpot VM performance FAQ for tips on profiling allocations (objects and their sizes) of your application:

<http://java.sun.com/docs/hotspot/index.html>

If your application runs out of maximum sessions as evidenced by a “too many active sessions” message in the server log file, and results in the container throwing exceptions, application performance will be impacted. To address the situation, consider the session manager properties, and the session idle time. Note that JSPs have sessions enabled by default.

Too Few Threads

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 Web Server, the server's default maximum threads setting is greater of 128 or the number of processors in the system. If you want your server to process more requests concurrently, you need to increase the maximum number of threads.

The symptom of a server with too few threads is a long response time. Making a request from a browser establishes a connection fairly quickly to the server, but if there are too few threads on the server it can take a long time before the response comes back to the client.

The best way to tell if your server is being throttled by too few threads is to see if the number of active sessions is close to, or equal to, the maximum number of threads. To do this, see [“Session Creation and Thread Information” on page 56](#).

Cache Not Utilized

If the file cache is not utilized, your server is not performing optimally. Since most sites have lots of GIF or JPEG files that are intended to 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 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, it is less important that the cache hit rate is low.

Check your hit ratio using statistics from `perfdump`, the Admin Console Monitoring tab, or `wadm stats` commands. 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 can even achieve 98% or higher. For more information, see [“File Cache Statistics Information” on page 58](#).

In addition, if you are doing a lot of CGI or NSAPI calls, you can have a low cache hit rate. If you have custom NSAPI functions, you can also have a low cache hit rate.

Keep-Alive Connections Flushed

A web site that can service 75 requests per second without keep-alive connections might 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 used effectively. If the `KeepAliveCount` shown in `perfdump` (Total Number of Connections Added, as displayed in the Admin Console) exceeds the keep-alive maximum connections, subsequent keep-alive connections are closed, or “flushed,” instead of being honored and kept alive.

Check the `KeepAliveFlushes` and `KeepAliveHits` values using statistics from `perfdump` or the Number of Connections Flushed and Number of Connections Processed under Keep Alive Statistics on the Monitoring Statistics page. For more information, see [“Keep-Alive Information” on page 51](#).

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 can.

To reduce keep-alive flushes, increase the keep-alive maximum connections. You can do this in the configuration's Performance Tab ⇒ HTTP sub tab or using the `wadm set-keep-alive props` command. The default is based on the number of available file descriptors in the system. By raising the keep-alive maximum connections value, you keep more waiting keep-alive connections open.



Caution – On UNIX/Linux systems, if the keep-alive maximum connections value is too high, the server can run out of open file descriptors. Typically 1024 is the limit for open files on UNIX/Linux, so increasing this value above 500 is not recommended.

Large Memory Footprint

Web Server automatically configures the connection queue size based on the number of available file descriptors in the system. The connection queue size on a system is determined by the sum total of `thread-pool/max-threads` element, `thread-pool/queue-size` element and `keep-alive/max-connections` element in the `server.xml` file.

For more information about the `server.xml` file, see the Administrator's Configuration File Reference.

In certain cases, the server's chosen defaults leads to larger memory footprint than what is required to run your applications. If the server selected defaults does not suit your needs, the memory usage of the server can be changed by specifying the values in `server.xml`. The `thread-pool/max-threads` is greater of 128 or the number of processors in the system unless explicitly specified in `server.xml`. The `thread-pool/queue-size` can be obtained from `perfdump` by examining the Connection Queue Information. For more information, see [“Connection Queue Information” on page 48](#). The `keep-alive/max-connections` can be obtained from [“Keep-Alive Information” on page 51](#) and [“Keep-Alive Count” on page 53](#). Logging at level `fine` will print these values in the error log file.

Log File Modes

Keeping the log files on a high-level of verbosity can have a significant impact on performance. On the configuration's General Tab ⇒ Log Settings page choose the appropriate log level and use levels such as `Fine`, `Finer`, and `Finest` with care. To set the log level using the CLI, use the command `wadm set-log-prop` and set the `log-level`.

Tuning of File Descriptors

Web Server 7.0 uses an algorithm to divide file descriptors to various needs on unix systems.

Here is the list of items which require file descriptors:

1. Web Applications.
Web Server leaves 80% file descriptors for web applications.
2. Daemon session threads connections.

For each daemon session thread Web Server 7.0 expects a average of 4 file descriptors. One client socket connection, requested file, an included file and a backend connection.

3. jdbc pools.
4. Access log counts for all virtual servers.
5. Listener counts.
6. File descriptors for file cache.
7. Keep alive file descriptors.
8. Thread pool queue size.

In above list (6), (7) and (8) are auto tuned which means that if user specifies them in `server.xml`, it uses those, otherwise it divides the remaining or available file descriptors to (6), (7) and (8). Available file descriptors = Total descriptors - item (1) to (5)

If available file descriptors are more than 1024 then Web Server uses 1:16:16 ratio for item (6),(7),(8). If available file descriptors are less than 1024 then it uses 1:16:8 ratio. File cache is given the least importance and keep alive is given the highest importance. It also rounds of the number to the power of 2.

Note that Web Server doesn't uses above algorithm for Windows systems. On Windows it uses 64K descriptors for keep alive, 16K descriptors for thread pool queue size.

Platform-Specific Issues and Tips

This chapter provides platform-specific tuning tips, and includes the following topics:

- “Solaris Platform-Specific Issues” on page 101
- “Solaris File System Tuning” on page 105
- “Solaris Platform-Specific Performance Monitoring” on page 106
- “Tuning Solaris for Performance Benchmarking” on page 108
- “Tuning UltraSPARC T1–Based Systems for Performance Benchmarking” on page 109

Solaris Platform-Specific Issues

This section discusses miscellaneous Solaris-specific issues and tuning tips, and includes the following topics:

- “Files Open in a Single Process (File Descriptor Limits)” on page 101
- “Failure to Connect to HTTP Server” on page 102
- “Connection Refused Errors” on page 103
- “Tuning TCP Buffering” on page 103
- “Using the Solaris Network Cache and Accelerator (SNCA)” on page 103

Files Open in a Single Process (File Descriptor Limits)

Different platforms each have limits on the number of files that can be open in a single process at one time. For busy sites, increase that number. On Solaris systems, control this limit by setting `rlim_fd_max` in the `/etc/system` file. For Solaris 8, the default is 1024, which you can increase to 65536. For Solaris 9 and 10, the default is 65536, which doesn't need to be increased.

After making this or any change in the `/etc/system` file, reboot Solaris to put the new settings into effect. In addition, if you upgrade to a new version of Solaris, remove any line added to `/etc/system` and add it again only after verifying that it is still valid.

An alternative way to make this change is using the `ulimit -n "value"` command. Using this command does not require a system restart. However, this command only changes the login shell, while editing the `etc/system` file affects all shells.

Failure to Connect to HTTP Server

If users are experiencing connection timeouts from a browser to Web Server when the server is heavily loaded, you can increase the size of the HTTP listener backlog queue. To increase this setting, edit the HTTP listener's listen queue value.

In addition to this setting, you must also increase the limits within the Solaris TCP/IP networking code. There are two parameters that are changed by executing the following commands:

```
/usr/sbin/ndd -set /dev/tcp tcp_conn_req_max_q 8192
/usr/sbin/ndd -set /dev/tcp tcp_conn_req_max_q0 8192
```

These two settings increase the maximum number of two Solaris listen queues that can fill up with waiting connections. `tcp_conn_req_max_q` increases the number of completed connections waiting to return from an `accept()` call. `tcp_conn_req_max_q0` increases the maximum number of connections with the handshake incomplete. The default values are 128 and 1024, respectively. To automatically have these `ndd` commands executed after each system reboot, place them in a file called `/etc/init.d/network-tuning` and create a link to that file named `/etc/rc2.d/S99network-tuning`.

You can monitor the effect of these changes by using the `netstat -s` command and looking at the `tcpListenDrop`, `tcpListenDropQ0`, and `tcpHalfOpenDrop` values. Review them before adjusting these values. If the parameters are not set to zero, adjust the value to 2048 initially, and continue to monitor the `netstat` output.

The Web Server HTTP listener's listen queue setting and the related Solaris `tcp_conn_req_max_q` and `tcp_conn_req_max_q0` settings are meant to match the throughput of the Web Server. These queues act as a "buffer" to manage the irregular rate of connections coming from web users. These queues allow Solaris to accept the connections and hold them until they are processed by the Web Server.

Do not accept more connections than the Web Server is able to process. Instead, limit the size of these queues and reject further connections than to accept excess connections and fail to service them. The value of 2048 for these three parameters typically reduces connection request failures, and improvement has been seen with values as high as 4096.

This adjustment is not expected to have any adverse impact in any web hosting environment, so you can consider this suggestion even if your system is not showing the symptoms mentioned.

Connection Refused Errors

If users are experiencing connection refused errors on a heavily loaded server, you can tune the use of network resources on the server.

When a TCP/IP connection is closed, the port is not reused for the duration of `tcp_time_wait_interval` (default value of 240000 milliseconds). This is to ensure that there are no leftover segments. The shorter the `tcp_time_wait_interval`, the faster precious network resources are again available. This parameter is changed by executing the following command. Do not reduce the parameter below 60000

```
usr/sbin/ndd -set /dev/tcp tcp_time_wait_interval 60000
```

To automatically have this `ndd` command executed after each system reboot, place it in a file called `/etc/init.d/network-tuning` and create a link to that file named `/etc/rc2.d/S99network-tuning`.

If your system is not exhibiting the symptoms mentioned, and if you are not well-versed in tuning the TCP protocol, do not change the above parameter.

Tuning TCP Buffering

If you are seeing unpredictable intermittent slowdowns in network response from a consistently loaded server, investigate setting the `sq_max_size` parameter by adding the following line to the `/etc/system` file:

```
set sq_max_size=512
```

This setting adjusts the size of the sync queue, which transfers packets from the hardware driver to the TCP/IP protocol driver. Using the value of 512 allows the queue to accommodate high volumes of network traffic without overflowing.

Using the Solaris Network Cache and Accelerator (SNCA)

The Solaris Network Cache and Accelerator (SNCA) is a caching server that provides improved web performance to the Solaris operating system.

It is assumed that SNCA has been configured for the system on which the Web Server is running. For more information about SNCA and its configuration and tuning, refer to the following man pages on your system:

- `ncab2clf(1)`
- `ncakmod(1)`

- `nca(1)`
- `snca(1)`
- `nca.if(4)`
- `ncakmod.conf(4)`
- `ncalogd.conf(4)`

▼ To Enable SNCA to Work With Web Server

This procedure assumes that SNCA has been configured, as discussed above.

- 1 From the **Common Tasks** page, choose a configuration and click **Edit Configuration**.
- 2 Click the **HTTP Listeners** tab and select the HTTP listener to edit.
- 3 On the **Edit HTTP Listener** page, set the **Protocol Family** to `nca`.
The HTTP listener must be listening on port 80 for this to work.
- 4 Save your changes.
- 5 Click the **Performance** tab.
- 6 Click the **Cache** sub tab.
- 7 On the **Cache Settings** page, make sure the file cache is enabled and enable **Use Sendfile**.
- 8 Save your changes.
- 9 Redeploy the configuration for your changes to take effect.

Maximum Threads and Queue Size

When configuring Web Server to be used with SNCA, disabling the thread pool provides better performance. These settings are on the configuration's **Performance** tab ⇒ **HTTP** sub tab, under **Thread Pool Settings**. To disable the thread pool, deselect the **Thread Pool Enabled** checkbox. You can also disable the thread pool using the `wadm set -thread-pool-prop` command's `enabled` property.

The thread pool can also be disabled with non-SNCA configurations, especially for cases in which short latency responses with no keep-alives must be delivered.

Solaris File System Tuning

This section discusses changes that can be made for file system tuning, and includes topics that address the following issues:

- [“High File System Page-In Rate” on page 105](#)
- [“Reduce File System Housekeeping” on page 105](#)
- [“Long Service Times on Busy Disks or Volumes” on page 105](#)

Read the descriptions of the following parameters carefully. If the description matches your situation, consider making the adjustment.

High File System Page-In Rate

If you are seeing high file system page-in rates on Solaris 8 or 9, increase the value of `segmap_percent`. This parameter is set by adding the following line to the `/etc/system` file:

```
set segmap_percent=25
```

`segmap_percent` adjusts the percentage of memory that the kernel maps into its address space for the file system cache. The default value is 12; that is, the kernel reserves enough space to map at most 12% of memory for the file system cache. On a heavily loaded machine with 4 GB of physical memory, improvements have been seen with values as high as 60. You can experiment with this value, starting with values around 25. On systems with large amounts of physical memory, you can raise this value in small increments, as it can significantly increase kernel memory requirements.

Reduce File System Housekeeping

UNIX file system (UFS) volumes maintain the time that each file was accessed. Note that the following change does not turn off the access time updates when the file is modified, but only when the file is accessed. If the file access time updates are not important in your environment, you can turn them off by adding the `noatime` parameter to the data volume's mount point in `/etc/vfstab`. For example:

```
/dev/dsk/c0t5d0s6 /dev/rdisk/c0t5d0s6 /data0 ufs 1 yes noatime
```

Long Service Times on Busy Disks or Volumes

Web Server's responsiveness depends greatly on the performance of the disk subsystem. Use the `iostat` utility to monitor how busy the disks are and how rapidly they complete I/O requests (the `%b` and `svc_t` columns, respectively). Service times are unimportant for disks that are less

than about 30% busy, but for busier disks, service times should not exceed about 20 milliseconds. If your busy disks have slower service times, improving disk performance can help Web Server performance substantially.

Your first step is to balance the load: if some disks are busy while others are lightly loaded, move some files off of the busy disks and onto the idle disks. If there is an imbalance, correcting it usually gives a far greater payoff than trying to tune the overloaded disks.

Solaris Platform-Specific Performance Monitoring

This section describes some of the Solaris-specific tools and utilities you can use to monitor your system's behavior, and includes the following topics:

- [“Short-Term System Monitoring” on page 106](#)
- [“Long-Term System Monitoring” on page 107](#)
- [““Intelligent” Monitoring” on page 107](#)

The tools described in this section monitor performance from the standpoint of how the system responds to the load that Web Server generates. For information about using Web Server's own capabilities to track the demands that users place on the Web Server itself, see [“Monitoring Server Performance” on page 20](#).

Short-Term System Monitoring

Solaris offers several tools for taking “snapshots” of system behavior. Although you can capture their output in files for later analysis, the tools listed below are primarily intended for monitoring system behavior in real time:

- The `iostat -x 60` command reports disk performance statistics at 60-second intervals.
Watch the `%b` column to see how much of the time each disk is busy. For any disk busy more than 20% of the time, pay attention to the service time as reported in the `svct` column. Other columns report the I/O operation rates, the amount of data transferred, and so on.
- The `vmstat 60` command summarizes virtual memory activity and some CPU statistics at 60-second intervals.
Monitor the `sr` column to keep track of the page scan rate and take action if it's too high. Note that “too high” is very different for Solaris 8 and 9 than for earlier releases. Watch the `us`, `sy`, and `id` columns to see how heavily the CPUs are being used; remember that you need to keep plenty of CPU power in reserve to handle sudden bursts of activity. Also keep track of the `r` column to see how many threads are contending for CPU time; if this remains higher than about four times the number of CPUs, reduce the server's concurrency.
- The `mpstat 60` command gives a detailed look at CPU statistics, while the `netstat -i 60` command summarizes network activity.

Long-Term System Monitoring

It is important not only to "spot-check" system performance with the tools mentioned above, but to collect longer-term performance histories so you can detect trends. If nothing else, a baseline record of a system performing well will help you figure out what has changed if the system starts behaving poorly. Enable the system activity reporting package by doing the following:

- Edit the file `/etc/init.d/perf` and remove the `#` comment characters from the lines near the end of the file. For Solaris 10, run the following command:
`svcadm enable system/sar`
- Run the command `crontab -e sys` and remove the `#` comment characters from the lines with the `sa1` and `sa2` commands. You can adjust how often the commands run and at what times of day depending on your site's activity profile. See the `crontab` man page for an explanation of the format of this file.

This command causes the system to store performance data in files in the `/var/adm/sa` directory, where by default they are retained for one month. You can then use the `sar` command to examine the statistics for time periods of interest.

"Intelligent" Monitoring

The SE toolkit is a freely downloadable software package developed by Sun performance experts. In addition to collecting and monitoring raw performance statistics, the toolkit can apply heuristics to characterize the overall health of the system and highlight areas that need adjustment. You can download the toolkit and its documentation from the following location:

<http://www.sunfreeware.com/setoolkit.html>

Solaris 10 Platform-Specific Tuning Information

DTrace is a comprehensive dynamic tracing framework for the Solaris Operating Environment. You can use the DTrace Toolkit to monitor the system. It is available from the following URL:

<http://www.opensolaris.org/os/community/dtrace/dtracetoolkit/>

Tuning Solaris for Performance Benchmarking

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

TABLE 4-1 Tuning Solaris for Performance Benchmarking

Parameter	Scope	Default Value	Tuned Value	Comments
rlim_fd_max	/etc/system	65536	65536	Process open file descriptors limit; accounts for the expected load (for the associated sockets, files, and pipes if any).
sq_max_size	/etc/system	2	0	Controls streams driver queue size; setting to 0 makes it infinite so the performance runs are not hit by lack of buffer space. Set on clients too. Note that setting sq_max_size to 0 is not be optimal for production systems with high network traffic.
tcp_time_wait_interval	ndd /dev/tcp	240000	60000	Set on clients too.
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%, 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 receive buffer.

Tuning UltraSPARC® T1–Based Systems for Performance Benchmarking

Use a combination of tunable parameters and other parameters to tune your system for performance benchmarking. These values are an example of how you can tune your system to achieve the desired result.

Tuning Operating System and TCP Settings

The following table shows the operating system tuning for Solaris 10 used when benchmarking for performance and scalability on UltraSPARC T1–based systems (64 bit systems).

TABLE 4–2 Tuning 64–bit Systems for Performance Benchmarking

Parameter	Scope	Default Value	Tuned Value	Comments
rlim_fd_max	/etc/system	65536	260000	Process open file descriptors limit; accounts for the expected load (for the associated sockets, files, pipes if any).
hires_tick	/etc/system		1	
sq_max_size	/etc/system	2	0	Controls streams driver queue size; setting to 0 makes it infinite so the performance runs are not hit by lack of buffer space. Set on clients too. Note that setting sq_max_size to 0 is not optimal for production systems with high network traffic.
ip:ip_squeue_bind			0	
ip:ip_squeue_fanout			1	
ipge:ipge_taskq_disable	/etc/system		0	
ipge:ipge_tx_ring_size	/etc/system		2048	
ipge:ipge_srv_fifo_depth	/etc/system		2048	
ipge:ipge_bcopy_thresh	/etc/system		384	
ipge:ipge_dvma_thresh	/etc/system		384	
ipge:ipge_tx_syncq	/etc/system		1	
tcp_conn_req_max_q	ndd /dev/tcp	128	3000	
tcp_conn_req_max_q0	ndd /dev/tcp	1024	3000	

TABLE 4–2 Tuning 64-bit Systems for Performance Benchmarking (Continued)

Parameter	Scope	Default Value	Tuned Value	Comments
tcp_max_buf	ndd /dev/tcp		4194304	
tcp_cwnd_max	ndd/dev/tcp		2097152	
tcp_xmit_hiwat	ndd /dev/tcp	8129	400000	To increase the transmit buffer.
tcp_rcv_hiwat	ndd /dev/tcp	8129	400000	To increase the receive buffer.

Note that the IPGE driver version is 1.25.25.

Disk Configuration

If HTTP access is logged, follow these guidelines for the disk:

- Write access logs on faster disks or attached storage.
- If running multiple instances, move the logs for each instance onto separate disks as much as possible.
- Enable the disk read/write cache. Note that if you enable write cache on the disk, some writes can be lost if the disk fails.
- Consider mounting the disks with the following options, which can yield better disk performance: `no logging`, `directio`, `noatime`.

Network Configuration

If more than one network interface card is used, make sure the network interrupts are not all going to the same core. Run the following script to disable interrupts:

```
allpsr='/usr/sbin/psrinfo | grep -v off-line | awk '{ print $1 }'
set $allpsr
numpsr=$#
while [ $numpsr -gt 0 ];
do
    shift
    numpsr='expr $numpsr - 1'
    tmp=1
    while [ $tmp -ne 4 ];
    do
        /usr/sbin/psradm -i $1
        shift
        numpsr='expr $numpsr - 1'
        tmp='expr $tmp + 1'
    done
done
```

Put all network interfaces into a single group. For example:

```
$ifconfig ipge0 group webserver  
$ifconfig ipge1 group webserver
```

Web Server Start Options

In some cases, performance can be improved by using large page sizes. To start the 32-bit Web Server with 4 MB pages:

```
LD_PRELOAD_32=/usr/lib/mpss.so.1 ; export LD_PRELOAD_32; export MPSSHEAP=4M;  
./bin/startserv; unset LD_PRELOAD_32; unset MPSSHEAP
```

For 64-bit servers:

```
LD_PRELOAD_64=/usr/lib/64/mpss.so.1; export LD_PRELOAD_64; export MPSSHEAP=4M;  
./bin/startserv; unset LD_PRELOAD_64; unset MPSSHEAP
```


Sizing and Scaling Your Server

This chapter examines the subsystems of your server, and provides recommendations for optimal performance. The chapter includes the following topics:

- “64-Bit Server” on page 113
- “Processors” on page 113
- “Memory” on page 114
- “Drive Space” on page 114
- “Networking” on page 114

64-Bit Server

The 64-bit server, available on Solaris SPARC and AMD64 platforms only, is more scalable than the 32-bit version. You can use the 64-bit server if your system has more than 4 GB of RAM. Some of the advantages which the 64-bit server has over the 32-bit server are:

- More file cache for static content
- Many simultaneous servlet sessions because of the 64-bit JVM

Processors

On Solaris and Windows, Web 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 CPU activity.

Memory

As a baseline, Web Server requires 64 MB RAM. Multiple CPUs require at least 64 MB for each CPU. For example, if you have four CPUs, install at least 256 MB RAM for optimal performance. For high numbers of peak concurrent users, also allow extra RAM for the additional threads. After the first 50 concurrent users, add an extra 512 KB for each peak concurrent user.

Drive Space

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

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

Your OS vendor can recommend how much swap or paging space to allocate. Based on testing, Web 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 can include multiple documents. If you are not sure, try using your home page and all of its associated subframes 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. The result is the WAN bandwidth your server needs.

For example, to support a peak of 50 users with an average document size of 24 KB, and to transfer each document in an average of 5 seconds, 240 KBs (1920 Kbit/s) are needed. Therefore, this site needs two T1 lines (each 1544 Kbit/s). This amount of bandwidth also allows some overhead for growth.

Your server's network interface card is intended to support more than the WAN to which it is connected. For example, if you have up to three T1 lines, one 10BaseT interface will be adequate. If you have 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 above to verify this.

Scalability Studies

This chapter describes the results of scalability studies. You can refer to these studies for a sample of how the server performs, and how you can configure your system to best take advantage of Web Server's strengths.

This chapter includes the following topics:

- “Study Goals” on page 115
- “Study Conclusion” on page 116
- “Hardware” on page 116
- “Software” on page 117
- “Configuration and Tuning” on page 117
- “Performance Tests and Results” on page 120

Study Goals

The goal of the tests in the study was to show how well Sun Java System Web Server 7 scales. The tests also helped to determine the configuration and tuning requirements for different types of content.

The studies were conducted with the following content:

- 100% static
- 100% C CGI
- 100% Perl CGI
- 100% NSAPI
- 100% Java servlets
- 100% PHP/FastCGI
- E-commerce web application with large inventory

Study Conclusion

When tuned, Sun Java System Web Server 7.0 scaled almost linearly in performance for dynamic and static content.

Hardware

With the exception of the e-commerce study, these studies were conducted using the following hardware. For hardware information for the e-commerce study, see [“Hardware for E-Commerce Test” on page 135](#).

Web Server system configuration for static content:

- Sun Microsystems Sun Fire T2000 (120 MHz, 8 cores) (only six cores were used for this test)
- 16256 Megabytes of memory
- Solaris 10 operating system
- Three Sun StoreEdge 3510

Web Server system configuration:

- Sun Microsystems Sun Fire T2000 (1000 MHz , 6 cores)
- 16376 Megabytes of memory
- Solaris 10 operating system

Driver system configuration:

- Three Sun Microsystems Sun Fire™ X4100
- Four Sun Microsystems Sun Fire V490 (2 X 1050 MHzUS-IV)
- Three Sun Fire T1000
- Sun Fire 880 (990 MHz US-III+)
- 8192 Megabytes of memory
- Solaris 10 operating system

Network configuration:

The Web Server and the driver machines were connected with multiple gigabit Ethernet links

Software

The load driver for these tests was an internally-developed Java application framework called the Faban driver.

Configuration and Tuning

The following tuning settings are common to all the tests in this study. Individual studies have additional configuration and tuning information.

/etc/system tuning:

```
set rlim_fd_max=500000
set rlim_fd_cur=500000
```

```
set sq_max_size=0
set consistent_coloring=2
set autoup=60
set ip:ip_queue_bind=0
set ip:ip_soft_rings_cnt=0
set ip:ip_queue_fanout=1
set ip:ip_queue_enter=3
set ip:ip_queue_worker_wait=0
```

```
set segmap_percent=6
set bufhwm=32768
set maxphys=1048576
set maxpgio=128
set ufs:smallfile=6000000
```

```
*For ipge driver
set ipge:ipge_tx_ring_size=2048
set ipge:ipge_tx_syncq=1
set ipge:ipge_srv_fifo_depth=16000
set ipge:ipge_reclaim_pending=32
set ipge:ipge_bcopy_thresh=512
set ipge:ipge_dvma_thresh=1
set pcie:pcie_aer_ce_mask=0x1
```

```
*For e1000g driver
set pcie:pcie_aer_ce_mask = 0x1
```

TCP/IP tuning:

```
ndd -set /dev/tcp tcp_conn_req_max_q 102400
ndd -set /dev/tcp tcp_conn_req_max_q0 102400
```

```
ndd -set /dev/tcp tcp_max_buf 4194304
ndd -set /dev/tcp tcp_cwnd_max 2097152
ndd -set /dev/tcp tcp_recv_hiwat 400000
ndd -set /dev/tcp tcp_xmit_hiwat 400000
```

Network Configuration

Since the tests use multiple network interfaces, it is important to make sure that all the network interfaces are not going to the same core. Network interrupts were enabled on one strand and disabled on the remaining three strand of a core using the following script:

```
allpsr='/usr/sbin/psrinfo | grep -v off-line | awk '{ print $1 }'
set $allpsr
numpsr=$#
while [ $numpsr -gt 0 ];
do
    shift
    numpsr='expr $numpsr - 1'
    tmp=1
    while [ $tmp -ne 4 ];
    do
        /usr/sbin/psradm -i $1
        shift
        numpsr='expr $numpsr - 1'
        tmp='expr $tmp + 1'
    done
done
```

The following example shows `psrinfo` output before running the script:

```
# psrinfo | more
0      on-line   since 12/06/2006 14:28:34
1      on-line   since 12/06/2006 14:28:35
2      on-line   since 12/06/2006 14:28:35
3      on-line   since 12/06/2006 14:28:35
4      on-line   since 12/06/2006 14:28:35
5      on-line   since 12/06/2006 14:28:35
.....
```

The following example shows `psrinfo` output after running the script:

```
0      on-line   since 12/06/2006 14:28:34
1      no-intr   since 12/07/2006 09:17:04
2      no-intr   since 12/07/2006 09:17:04
3      no-intr   since 12/07/2006 09:17:04
4      on-line   since 12/06/2006 14:28:35
```

```
5      no-intr    since 12/07/2006 09:17:04
      .....

```

Web Server Tuning

The following table shows the tuning settings used for the Web Server.

TABLE 6-1 Web Server Tuning Settings

Component	Default	Tuned
Access logging	enabled=true	enabled=false
Thread pool	min-threads=16 max-threads=128 stack-size=131072 queue-size=1024	min-threads=128 max-threads=200 stack-size=262144 queue-size=15000
HTTP listener	Non-secure listener on port 80 listen-queue-size=128	Non-secure listener on port 80 Secure listener on port 443 listen-queue-size=15000
Keep alive	enabled=true threads=1 max-connections=200 timeout=30 sec	enabled=true threads=2 max-connections=15000 timeout=180 sec
default-web.xml	JSP compilation turned on	JSP compilation turned off

The following table shows the SSL session cache tuning settings used for the SSL tests.

TABLE 6-2 SSL Session Cache Tuning Settings

Component	Default
SSL session cache	enabled=true max-entries=10000 max-ssl2-session-age=100 max-ssl3-tls-session-age=86400

Performance Tests and Results

This section contains the test-specific configuration, tuning, and results for the following tests:

- “Static Content Test” on page 120
- “Dynamic Content Test: Servlet” on page 122
- “Dynamic Content Test: C CGI” on page 123
- “Dynamic Content Test: Perl CGI” on page 125
- “Dynamic Content Test: NSAPI” on page 126
- “PHP Scalability Tests” on page 127
- “SSL Performance Test: Static Content” on page 130
- “SSL Performance Test: Perl CGI” on page 132
- “SSL Performance Test: C CGI” on page 133
- “SSL Performance Test: NSAPI” on page 134
- “E-Commerce Web Application Test” on page 135

The following metrics were used to characterize performance:

- Operations per second (ops/sec) = successful transactions per second
- Response time for single transaction (round-trip time) in milliseconds

The performance and scalability diagrams show throughput (ops/sec) against the number of cores enabled on the system.

Static Content Test

This test was performed with a static download of a randomly selected file from a pool of 10,000 directories, each containing 36 files ranging in size from 1KB to 1000 KB. The goal of the static content test was to saturate the cores and find out the respective throughput and response time.

This test used the following configuration:

- Static files were created on striped disk array (Sun StorEdge 3510).
- Multiple network interfaces were configured.
- Web Server was configured with 64 bit.
- File-cache was enabled with the tuning settings described in the following table.

TABLE 6-3 File Cache Configuration

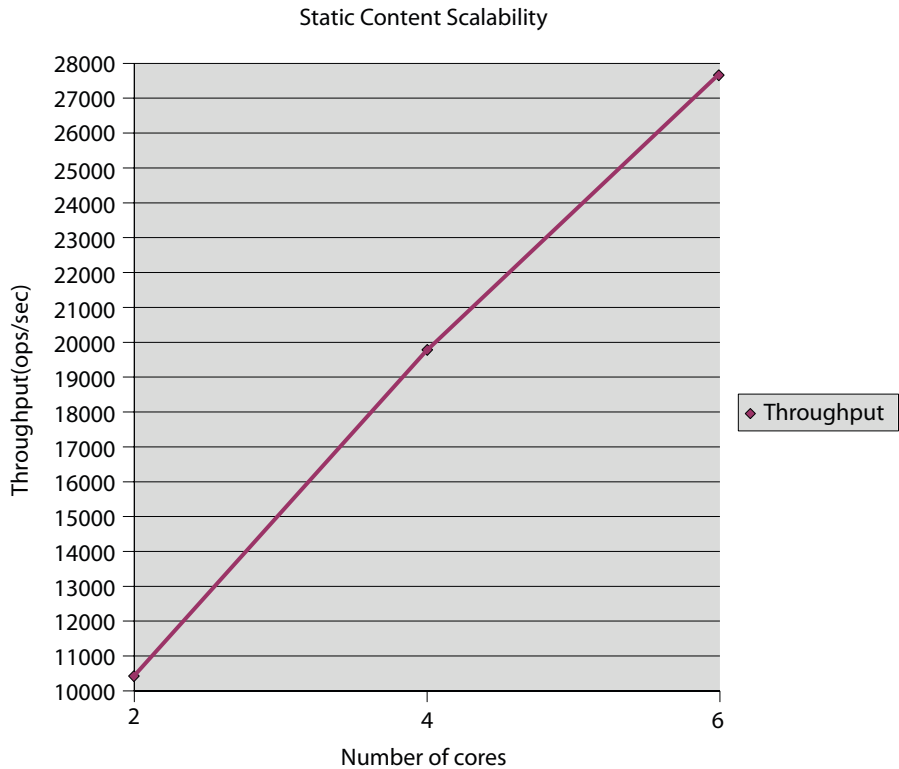
Default	Tuned
enabled=true	enabled=true
max-age=30 sec	max-age=3600
max-entries=1024	max-entries=1048576
sendfile=false	sendfile=true
max-heap-file-size=524288	max-heap-file-size=1200000
max-heap-space=10485760	max-heap-space=8000000000
max-mmap-file-size=0	max-mmap-file-size=1048576
max-mmap-space=0	max-mmap-space=1
	max-open-files=1048576

The following table shows the static content scalability results.

TABLE 6-4 Static Content Scalability

Number Of Cores	Average Throughput (ops/sec)	Average Response Time (ms)
2	10365	184
4	19729	199
6	27649	201

The following is a graphical representation of static content scalability results.



Dynamic Content Test: Servlet

This test was conducted using the servlet. The test prints out the servlet's initialization arguments, environments, request headers, connection and client information, URL information, and remote user information. JVM tuning settings were applied to the server. The goal was to saturate the cores on the server and find out the respective throughput and response time.

The following table shows the JVM tuning settings used in the test.

TABLE 6-5 JVM Tuning Settings

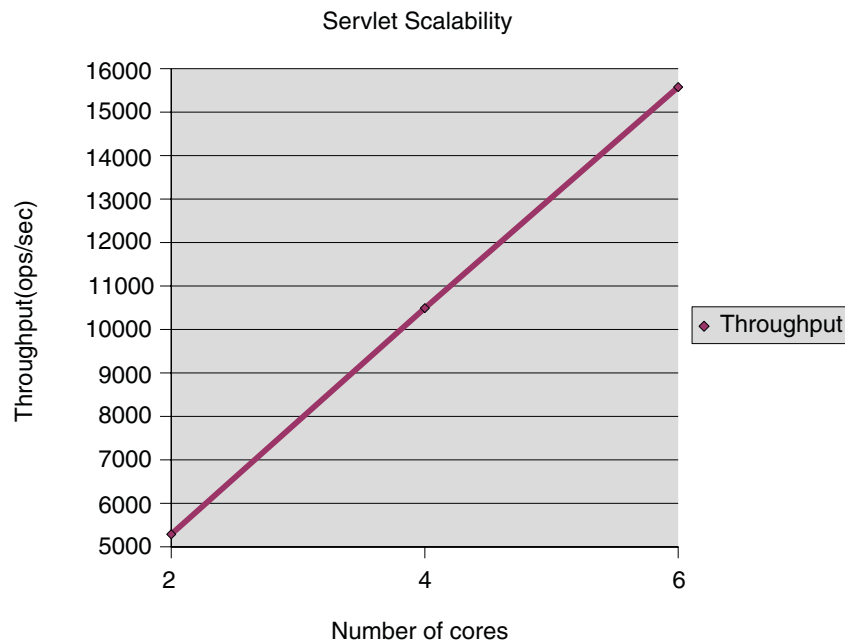
Default	Tuned
-Xmx128m	-server -Xrs -Xmx2048m -Xms2048m -Xmn2024m
-Xms256m	-XX:+AggressiveHeap -XX:LargePageSizeInBytes=256m
	-XX:+UseParallelOldGC -XX:+UseParallelGC
	-XX:ParallelGCThreads=<number of cores>
	-XX:+DisableExplicitGC

The following table shows the results for the dynamic content servlet test.

TABLE 6-6 Dynamic Content Test: Servlet Scalability

Number Of Cores	Average Throughput (ops/sec)	Average Response Time (ms)
2	5287	19
4	10492	19
6	15579	19

The following is a graphical representation of servlet scalability results.



Dynamic Content Test: C CGI

This test was performed by accessing a C executable called `printenv`. This executable outputs the environment variable information. CGI tuning settings were applied to the server. The goal was to saturate the cores on the server and find out the respective throughput and response time.

The following table describes the CGI tuning settings used in this test.

TABLE 6-7 CGI Tuning Settings

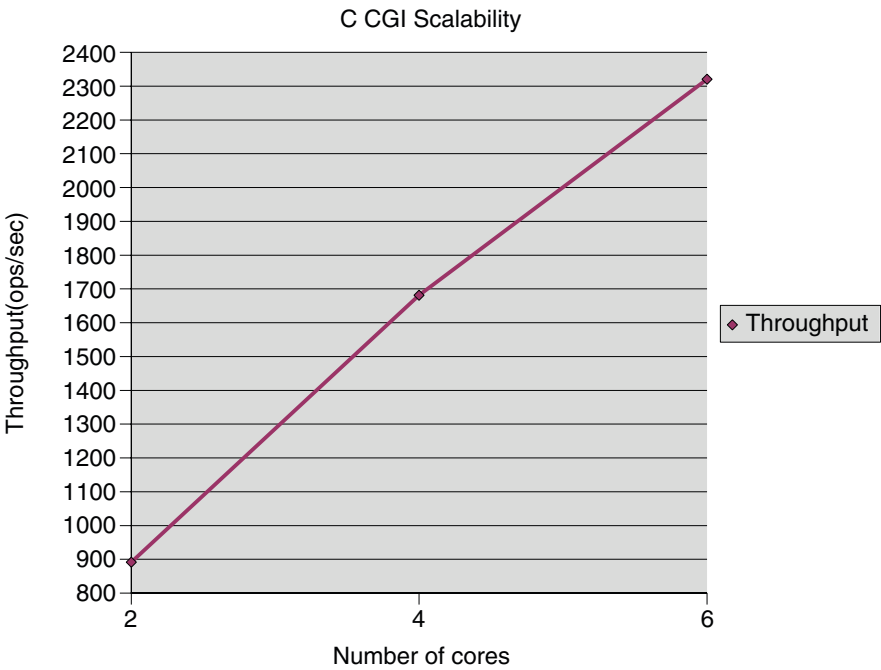
Default	Tuned
idle-timeout=300	idle-timeout=300
cgistub-idle-timeout=30	cgistub-idle-timeout=1000
min-cgistubs=0	min-cgistubs=100
max-cgistubs=16	max-cgistubs=100

The following table shows the results of the dynamic content test for C CGI.

TABLE 6-8 Dynamic Content Test: C CGI Scalability

Number Of Cores	Average Throughput (ops/sec)	Average Response Time (ms)
2	892	112
4	1681	119
6	2320	129

The following is a graphical representation of C CGI scalability results.



Dynamic Content Test: Perl CGI

This test was conducted with Perl script called `printenv.pl` that prints the CGI environment. CGI tuning settings were applied to the server. The goal was to saturate the cores on the server and find out the respective throughput and response time.

The following table shows the CGI tuning settings used in the dynamic content test for Perl CGI.

TABLE 6-9 CGI Tuning Settings

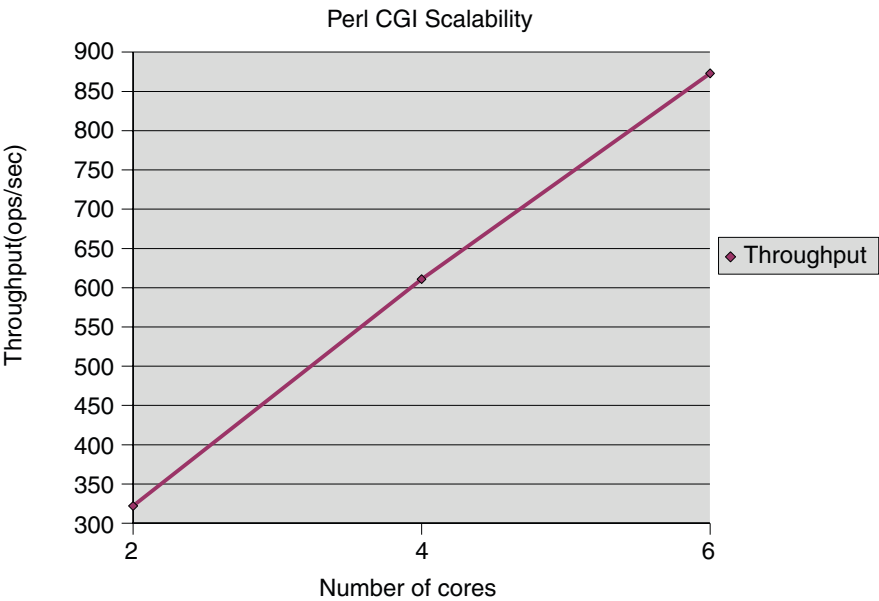
Default	Tuned
idle-timeout=300	idle-timeout=300
cgistub-idle-timeout=30	cgistub-idle-timeout=1000
min-cgistubs=0	min-cgistubs=100
max-cgistubs=16	max-cgistubs=100

The following table shows the results for the dynamic content test of Perl CGI.

TABLE 6-10 Dynamic Content Test: Perl CGI Scalability

Number Of Cores	Average Throughput (ops/sec)	Average Response Time (ms)
2	322	310
4	611	327
6	873	343

The following is a graphical representation of Perl CGI scalability results.



Dynamic Content Test: NSAPI

The NSAPI module used in this test was `printenv2.so`. It prints the NSAPI environment variables along with some text to make the entire response 2 KB. The goal was to saturate the cores on the server and find out the respective throughput and response time.

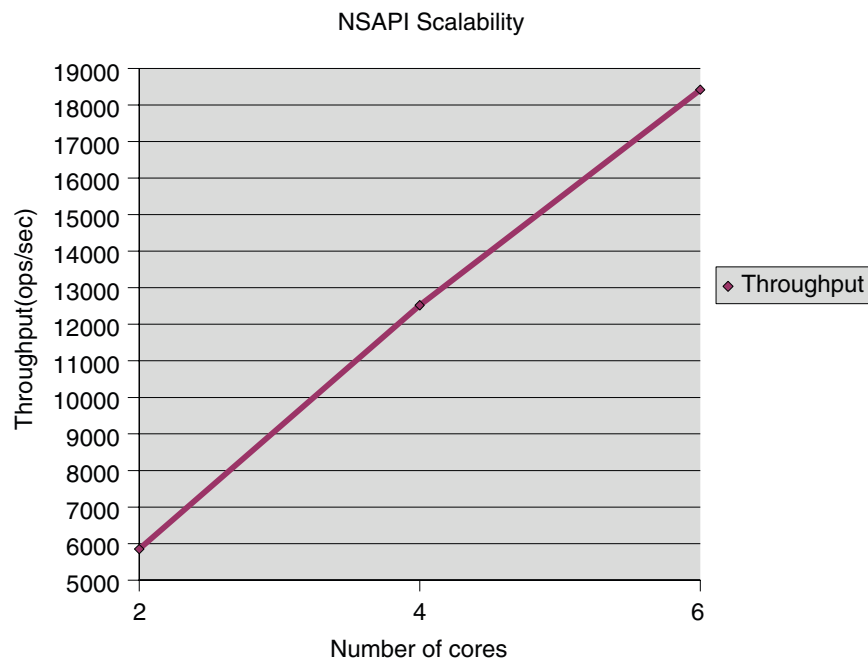
The only tuning for this test was optimizing the path checks in `obj.conf` by removing the unused path checks.

The following table shows the results of the dynamic content test for NSAPI.

TABLE 6-11 Dynamic Content Test: NSAPI Scalability

Number Of Cores	Average Throughput (ops/sec)	Average Response Time (ms)
2	6264	14
4	12520	15
6	18417	16

The following is a graphical representation of NSAPI scalability results.



PHP Scalability Tests

PHP is a widely-used scripting language uniquely suited to creating dynamic, Web-based content. It is the most rapidly expanding scripting language in use on the Internet due to its simplicity, accessibility, wide number of available modules, and large number of easily available applications.

The scalability of Web Server combined with the versatility of the PHP engine provides a high-performing and versatile web deployment platform for dynamic content. These tests used PHP version 5.1.6.

The tests were performed in two modes:

- An out-of-process `fastcgi-php` application invoked using the FastCGI plug-in.
- In-process PHP NSAPI plug-in.

The test executed the `phpinfo()` query. The goal was to saturate the cores on the server and find out the respective throughput and response time.

PHP Scalability with Fast CGI

The following table shows the Web Server tuning settings used for the FastCGI plug-in test

TABLE 6-12 Tuning Settings for FastCGI Plug-in Test

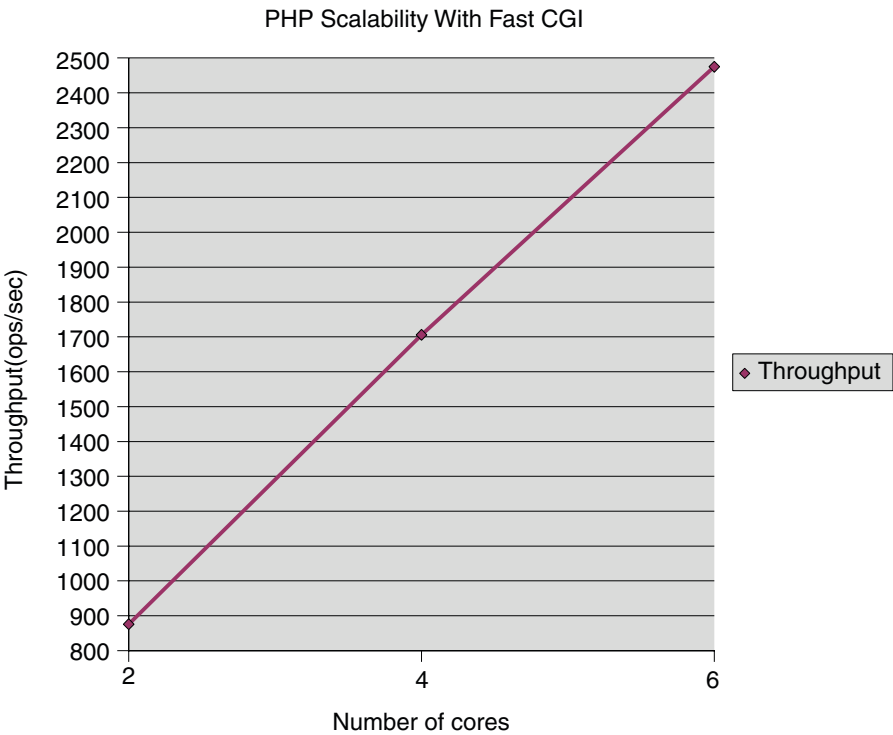
Configuration	Tuning
magnus.conf	Init fn="load-modules" shlib="path_to_web_server_plugin_dir/fastcgi/libfastcgi.so" funcs="responder_fastcgi" shlib_flags="(global now)"
obj.conf	NameTrans fn="assign-name" from="/fcgi/*" name="fcgi.config" <Object name="fcgi.config"> Service type="magnus-internal/ws-php" fn="responder-fastcgi" app-path="path_to_php" bind-path="localhost:9000" app-env="PHP_FCGI_CHILDREN=128" app-env="PHP_FCGI_MAX_REQUESTS=20000" app-env="LD_LIBRARY_PATH=path_to_php_lib" listen-queue=8192 req-retry=2 reuse-connection=1 connection-timeout=120 resp-timeout=60 restart-interval=0 </Object>
mime.types	type=magnus-internal/ws-php exts=php,php3,php4

The following table shows the results of the PHP with FastCGI test.

TABLE 6-13 PHP Scalability with Fast CGI

Number of Cores	Average Throughput (ops/sec)	Average Response Time (ms)
2	876	114
4	1706	117
6	2475	121

The following is a graphical representation of PHP scalability with Fast CGI.



PHP Scalability with NSAPI

The following table shows the Web Server tuning settings for the PHP with NSAPI test.

TABLE 6-14 NSAPI Plug-in Configuration for PHP

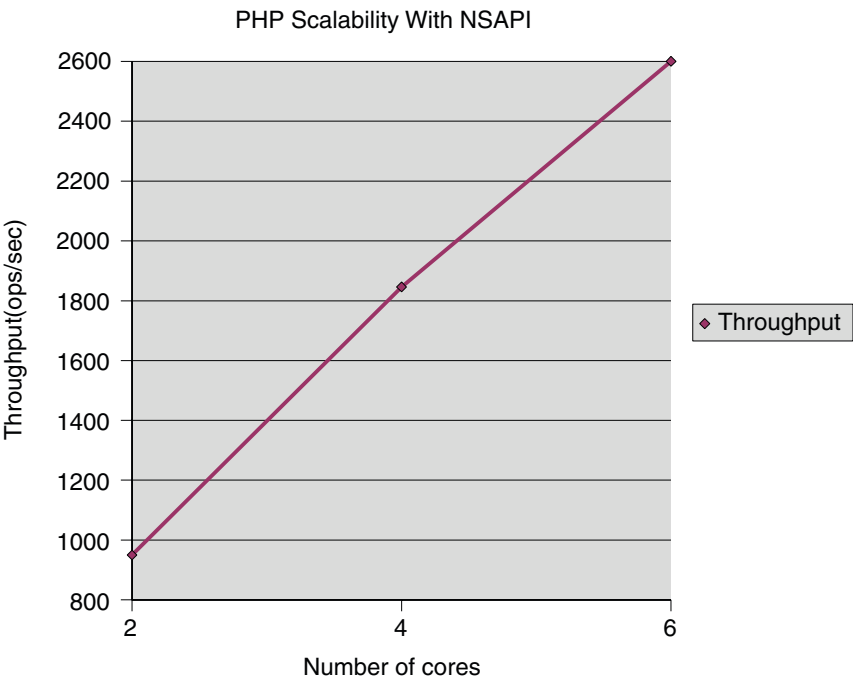
magnus.conf	<pre>Init fn="load-modules" shlib="libphp5.so" funcs="php5_init,php5_close,php5_execute" Init fn="php5_init" errorString="PHP Totally Blew Up!"</pre>
obj.conf	<pre>NameTrans fn="pfx2dir" from="/php-nsapi" dir="path_to_php_script_dir" name="php-nsapi" <Object name="php-nsapi"> ObjectType fn="force-type" type="magnus-internal/x-httpd-php" Service fn=php5_execute </Object></pre>
mime.types	<pre>type=magnus-internal/ws-php exts=php,php3,php4</pre>

The following table shows the results of the PHP with NSAPI test.

TABLE 6-15 PHP Scalability with NSAPI

Number of Cores	Average Throughput (ops/sec)	Average Response Time (ms)
2	950	105
4	1846	108
6	2600	115

The following is a graphical representation of PHP scalability with NSAPI.



SSL Performance Test: Static Content

This test was performed with static download of a randomly selected file from a pool of 10,000 directories, each containing 36 files ranging in size from 1KB to 1000 KB. The goal of the SSL static content tests was to saturate the cores and find out the respective throughput and response time. Only four cores of T2000 were used for this test.

This test used the following configuration:

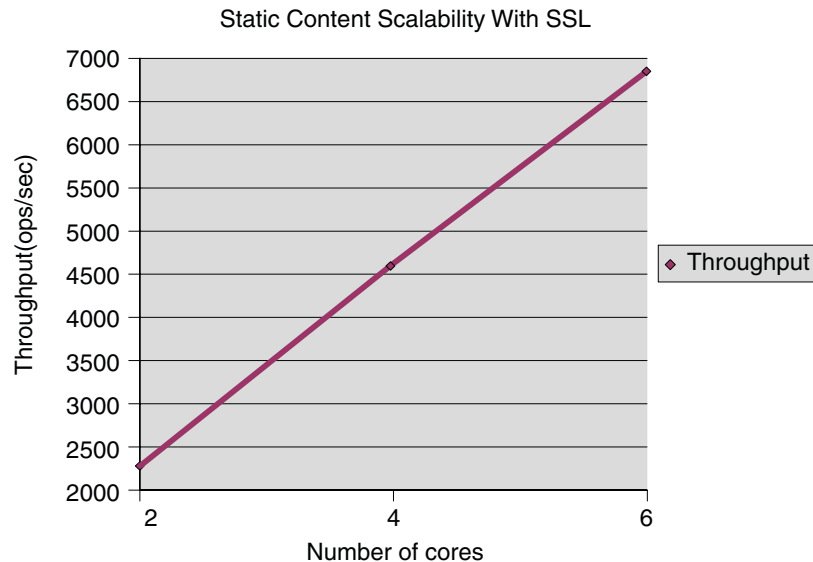
- Static files were created on striped disk array (Sun StorEdge 3510).
- Multiple network interfaces were configured.
- The file cache was enabled and tuned using the settings in [Table 6-3](#).
- The SSL session cache was tuned using the settings in [Table 6-2](#).
- Web Server is configured with 64 bit

The following table shows the SSL static content test results.

TABLE 6-16 SSL Performance Test: Static Content Scalability

Number of Cores	Average Throughput (ops/sec)	Average Response Time (ms)
2	2284	379
4	4538	387
6	6799	387

The following is a graphical representation of static content scalability with SSL.



SSL Performance Test: Perl CGI

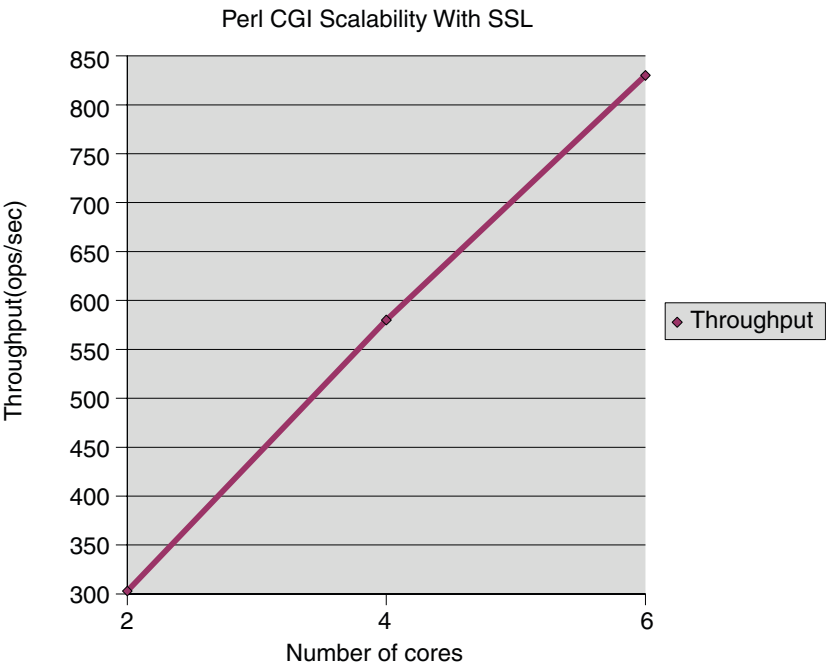
This test was conducted with Perl script called `printenv.pl` that prints the CGI environment in SSL mode. The test was performed in SSL mode with the SSL session cache enabled. The goal was to saturate the cores on the server and find out the respective throughput and response time.

The following table shows the SSL Perl CGI test results.

TABLE 6-17 SSL Performance Test: Perl CGI Scalability

Number of Cores	Average Throughput (ops/sec)	Average Response Time (ms)
2	303	329
4	580	344
6	830	361

The following is a graphical representation of Perl scalability with SSL.



SSL Performance Test: C CGI

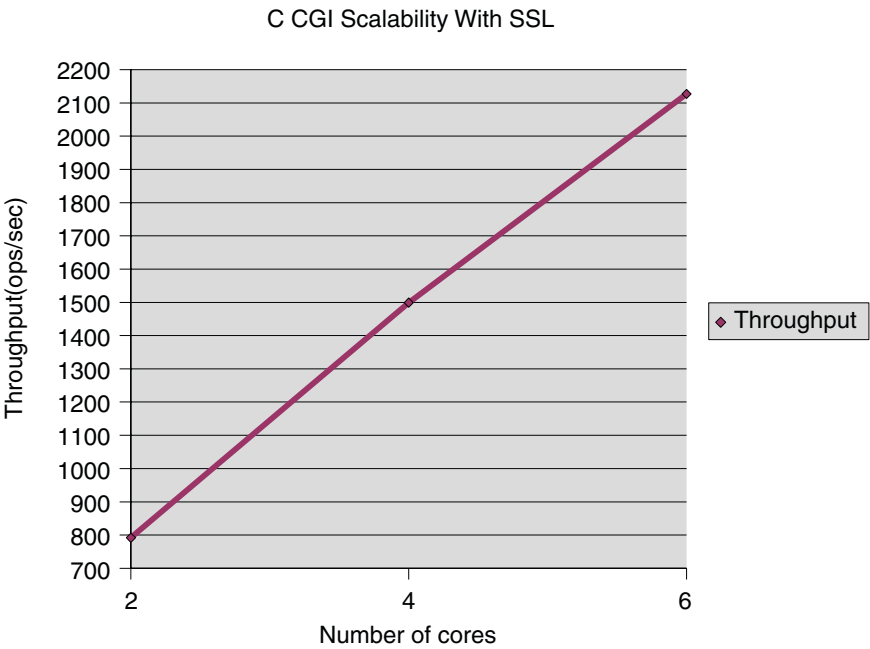
This test was performed by accessing a C executable called `printenv` in SSL mode. This executable outputs the environment variable information. The test was performed in SSL mode with the SSL session cache enabled. The goal was to saturate the cores on the server and find out the respective throughput and response time.

The following table shows the SSL CGI test results.

TABLE 6-18 SSL Performance Test: C CGI Scalability

Number of Cores	Average Throughput (ops/sec)	Average Response Time (ms)
2	792	126
4	1499	133
6	2127	141

The following is a graphical representation of C CGI scalability with SSL.



SSL Performance Test: NSAPI

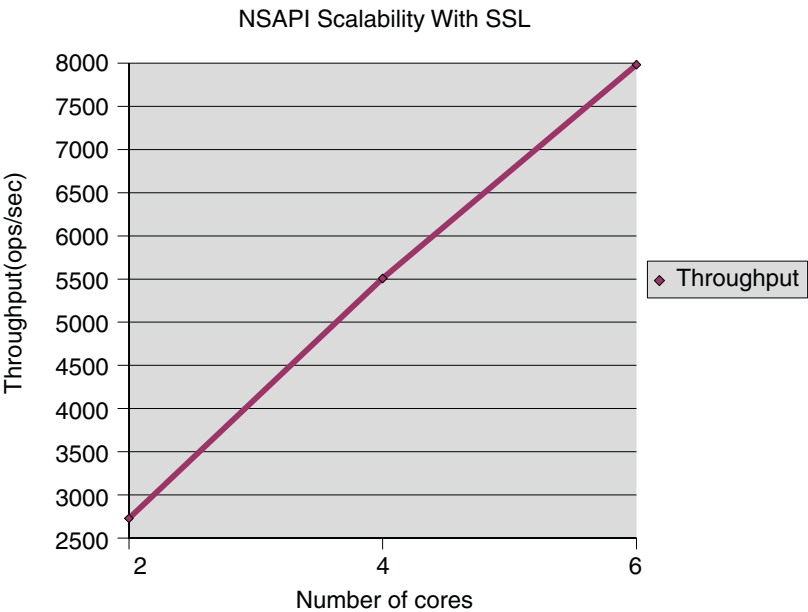
The NSAPI module used in this test was `printenv2`. so. It prints the NSAPI environment variables along with some text to make the entire response 2 KB. The test was performed in SSL mode with the SSL session cache enabled. The goal was to saturate the cores on the server and find out the respective throughput and response time.

The following table shows the SSL NSAPI test results.

TABLE 6-19 SSL Performance Test: NSAPI Scalability

Number of Cores	Average Throughput (ops/sec)	Average Response Time (ms)
2	2729	29
4	5508	30
6	7982	32

The following is a graphical representation of NSAPI scalability with SSL.



E-Commerce Web Application Test

The e-commerce application is a more complicated application that utilizes a database to simulate online shopping.

Hardware for E-Commerce Test

The e-commerce studies were conducted using the following hardware.

Web Server system configuration:

- Sun Microsystems Sun Fire 880 (900MHz US-III+). Only four CPUs were used for this test.
- 16384 Megabytes of memory.
- Solaris 10 operating system.

Database system configuration:

- Sun Microsystems Sun Fire 880 (900MHz US-III+)
- 16384 Megabytes of memory
- Solaris 10 operating system
- Oracle 10.1.0.2.0

Driver system configuration:

- Sun Microsystems Sun Fire 880 (900MHz US-III+)
- Solaris 10 operating system

Network configuration:

The Web Server, database, and the driver machines were connected with a gigabit Ethernet link.

Configuration and Tuning for E-Commerce Test

The e-commerce test was run with the following tuning settings.

JDBC tuning:

```
<jdbc-resource>
  <jndi-name>jdbc/jwebapp</jndi-name>
  <datasource-class>oracle.jdbc.pool.OracleDataSource</datasource-class>
  <max-connections>200</max-connections>
  <idle-timeout>0</idle-timeout>
  <wait-timeout>5</wait-timeout>
  <connection-validation>auto-commit</connection-validation>
  <property>
    <name>username</name>
    <value> db_user  </value>
  </property>
</jdbc-resource>
```

```
<property>
  <name>password</name>
  <value> db_password    </value>
</property>
<property>
  <name>url</name>
  <value>jdbc:oracle:thin:@db_host_name:1521:oracle_sid</value>
</property>
<property>
  <name>ImplicitCachingEnabled</name>
  <value>true</value>
</property>
<property>
  <name>MaxStatements</name>
  <value>200</value>
</property>
</jdbc-resource>
```

JVM tuning:

```
-server -Xmx1500m -Xms1500m -Xss128k -XX:+DisableExplicitGC
```

E-commerce Application Description

This test models an e-commerce web site that sells items from a large inventory. It uses the standard web application model-view-controller design pattern for its implementation: the user interface (that is, the view) is handled by 16 different JSP pages which interface with a single master control servlet. The servlet maintains JDBC connections to the database, which serves as the model and handles 27 different queries. The JSP pages make extensive use of JSP tag libraries and comprise almost 2000 lines of logic.

Database Cardinality

The database contains 1000 orderable items (which have two related tables which also have a cardinality of 1000), 72000 customers (with two related tables), and 1.9 million orders (with two related tables). Standard JDBC connections handle database connection using prepared statements and following standard JDBC design principles.

Workload

A randomly-selected user performs the online shopping. The following operations were used in the Matrix mix workload. Operations were carried out with the following precedence: Home, AdminConfirm, AdminRequest, BestSellers, BuyConfirm, BuyRequest, CustomerRegistration, NewProducts, OrderDisplay, OrderInquiry, ProductDetail, SearchRequest, SearchResults, and ShoppingCart.

The Faban driver was used to drive the load. Think time was chosen from a negative exponential distribution. The minimum think time was 7.5 seconds, the maximum was 75 seconds. The maximum number of concurrent users that the system can support was based on the following passing criteria.

TABLE 6-20 Performance Test Pass Criteria

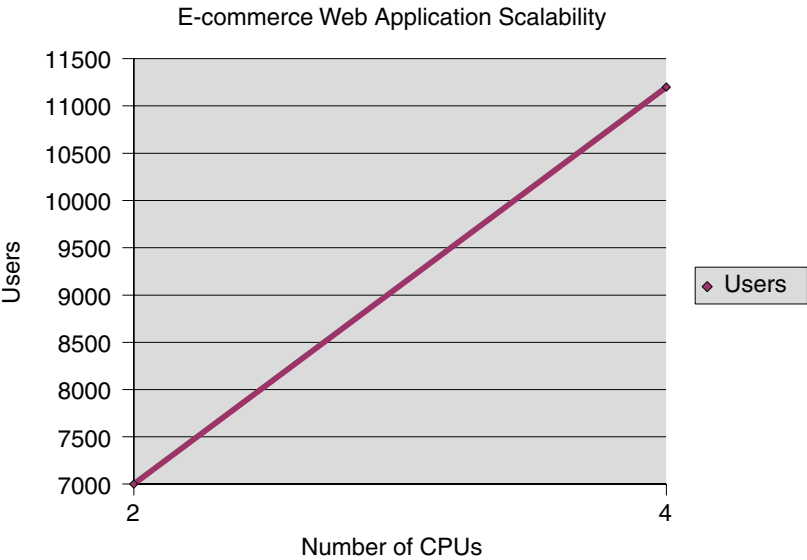
Transaction	90th Percentile Response Time (Seconds)
HomeStart	3
AdminConfirm	20
AdminRequest	3
BestSellers	5
BuyConfirm	5
BuyRequest	3
CustomerRegistration	3
Home	3
NewProducts	5
OrderDisplay	3
OrderInquiry	3
ProductDetail	3
SearchRequest	3
SearchResults	10
ShoppingCart	3

The following table shows the e-commerce web application test results.

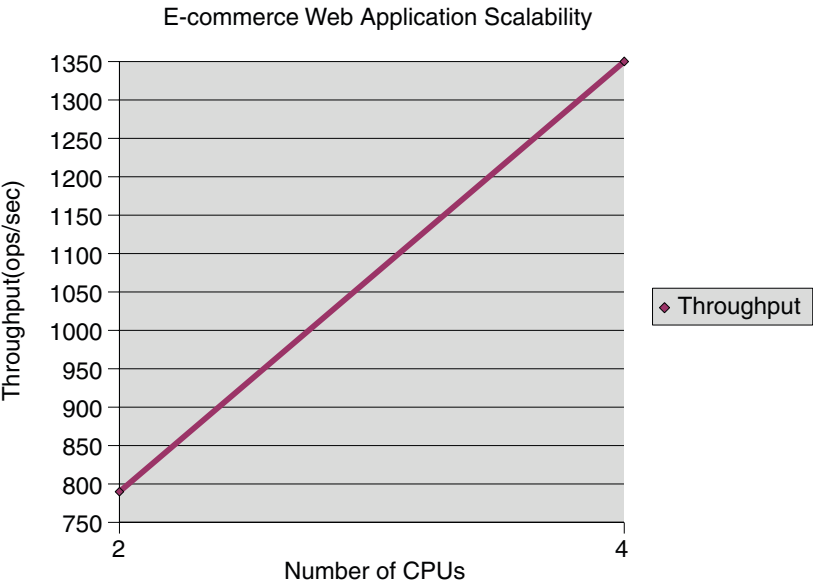
TABLE 6-21 E-Commerce Web Application Scalability

Number of CPUs	Users	Throughput (ops/sec)
2	7000	790
4	11200	1350

The following is a graphical representation of e-commerce web application scalability.



The following is a graphical representation of e-commerce web application scalability.



Index

Numbers and Symbols

- 64-bit servers
 - performance advantages, 19
 - scaling, 113

A

- acceptor threads, 51
- access time updates, 105
- acl-bucket, 34
- ACL user cache, 77-78
 - max-groups-per-user, 77
 - max-users, 77
 - maximum age, 77
- activating statistics, 22-23
- AddLog, 83
- Admin Console, more information about, 12
- assign-name function, 81, 82
- async DNS cache, 70
- auto-commit validation method, 76

B

- benchmarking
 - tuning Solaris for, 108, 109-110
- bottleneck, ACL user cache, 77
- buckets, performance, 33
- busy functions, 83

C

- cache not utilized, 97
- caching, servlet/JSP, 79
- cgi-bucket, 34
- CGIStub processes, 81
- check-acl SAF, 95
- class-loader, 79
- class reloading, configuring, 79
- classpath, directories in, 79-80
- configurations
 - and performance, 18
 - statistics, 21
- connection handling, 38-40
- connection queue information, 48-50
- connection queue size, and SNCA, 104
- connection refused errors, 103
- connection settings, JDBC resource, 75-77
- connection timeouts, 102
- connections, 38-44
 - closed, 53
 - JDBC, 73
 - simultaneous, 57
 - simultaneous using maximum threads setting, 43
- content_length header, 53
- crontab -e sys command, 107

D

- default-bucket, 33
- determining requirements, 114
- directories in the classpath, 79-80

- disabling network interrupts, 110-111
- disk configuration, 110
- DNS cache, 68-70
 - async enabled, 70
 - current entries, 69
 - entries, 69
 - hit ratio, 69
 - maximum entries, 69
- drive space, sizing issues, 114
- dynamic control and monitoring, file cache, 63
- dynamic reload interval, 79

E

- enable-perfdump command, 30
- enable-stats-xml command, 28
- enabling statistics, 22-23
- etc/system file, 101
 - in scalability studies, 117

F

- Faban driver, 117
- fail all connections, JDBC resource, 76
- file-bucket, 34
- file cache, 58-65
 - cache lookups, 61-62
 - dynamic control and monitoring, 63
 - entries, 61
 - flags for ?list option, 65
 - hit ratio, 61-62
 - low hit rate with custom NSAPI functions, 97
 - maximum age, 62
 - maximum heap size, 62
 - nocache parameter, 62
 - obj.conf object for monitoring, 63
 - problems, cache not utilized, 97
 - status example, 64
- file system tuning, Solaris, 105-106
- find-pathinfo-forward, 81-82
- find-pathinfo function, 81
- flushed keep-alive connections, 97-98
- free connections, in JDBC resources, 74

- func_insert, 83

G

- get-config-stats command, 24
- get-perfdump command, 31
- get-stats-xml command, 29
- get-virtual-server-stats command, 26
- get-webapp-stats command, 26
- guarantee isolation, JDBC resource, 77

H

- hardware
 - for e-commerce study, 135
 - for studies, 116
- high concurrency mode, 39
- high file system page-in rate, 105
- hires_tick, 109
- hit ratio, 61, 97
- HotSpot VM performance FAQ, 96
- HTTP 1.0-style workload, 55
- HTTP 1.1-style workload, 55-56
- HTTP access logged, 110
- HTTP listener, statistics, 50

I

- idle threads, 66
- idle timeout
 - JDBC resource, 75
- init-cgi, multi-process mode, 81
- iostat -x 60 command, 106
- iostat utility, 105
- ip:ip_squeue_bind, 109
- ip:ip_squeue_fanout, 109
- ipge:ipge_bcopy_thresh, 109
- ipge:ipge_srv_fifo_depth, 109
- ipge:ipge_taskq_disable, 109
- ipge:ipge_tx_ring_size, 109
- ipge:ipge_tx_syncq, 109

J

- Java ES monitoring console, 35
- Java heap tuning, 71
- Java HotSpot VM, 71
- java.lang.OutOfMemoryError, 96
- Java Security Manager, configuring, 79
- Java VM heap space, 96
- Java web applications, tuning performance, 78-81
- JDBC connection pooling, improving application performance, 72
- JDBC resources, 72-77
 - connection settings, 75-77
 - connections, 73
 - free connections, 74
 - idle timeout, 75
 - leased connections, 74
 - maxConnections, 74
 - peakConnections, 75
 - queued connections, 74
 - statistics in Admin Console, 73-74
 - validation method, 76
- jsp-config, 78
- JVM, 70-71
 - Java heap tuning, 71

K

- keep-alive, 51-56
 - connections flushed, 97-98
 - count, 53-54, 97
 - flushes, 54, 97
 - hits, 54, 97
 - maximum connections, 53, 97
 - maximum number of connections, 53
 - poll interval, 54
 - refusals, 54
 - threads, 55
 - timeout, 53
 - timeouts, 54
- KernalThreads directive, 40

L

- LateInit, 81
- LDAP server, and ACL user cache, 77
- leased connections, in JDBC resources, 74
- listen socket, statistics, 50
- load driver, for studies, 117
- load-modules function, 42
- log file modes, 98
 - verbose, 98
- long service times, 105-106
- low latency mode, 39
- low-memory problems, 96

M

- magnus.conf
 - connection-handling directives, 40
 - init-cgi, multi-process mode, 81
- manager-properties properties, 80
- max-groups-per-user, ACL user cache, 77
- max-users, ACL user cache, 77
- maximum age, file cache, 62
- maximum connections, JDBC resource, 75
- maximum heap size, 62
- maximum threads, 43, 57, 96
 - and NativePoolQueueSize, 67
 - and SNCA, 104
 - too few threads, 96
- maxLocks, tuning, 80
- MaxProcs, 43, 44
- maxSessions, 80
- memory, sizing issues, 114
- memory requirements, 114
- meta-data validation method, 76
- minimum connections, JDBC resource, 75
- MMapSessionManager, tuning, 80-81
- modes
 - log file, 98
 - multi-process, 43-44
 - single-process, 42-43
- monitoring server performance
 - comparison of methods, 20
 - methods with least impact, 20
 - overview, 17-35

monitoring server performance (*Continued*)

- using Java EE monitoring console, 35
 - using perfdump, 29-35
 - using performance buckets, 33-35
 - using SE toolkit, 107
 - using stats-xml, 27-29
- mpstat 60 command, 106
- multi-process mode, 42-44

N

- NameTrans, 42, 81, 82
- native thread pool, 41-42, 65-66
- NativePoolMaxThreads, 66, 67-68
- NativePoolMinThreads, 68
- NativePoolQueueSize, 66, 67
- NativePoolStackSize, 66-67
- NativeThread, 42
- ndd command, 103
- netstat -i 60, 106
- netstat -s command, 102
- network configuration, 110-111
- for studies, 118-119
- network interrupts, disabling, 110-111
- networking, sizing issues, 114
- nocache parameter, 62
- nostat, 82-83
- NSPR, 41
- NSServletService, 33
- ntrans-base, 81, 82

O

- obj.conf
- custom thread pool, 40
 - object for monitoring the file cache, 63
 - performance buckets, 34
 - UseOutputStreamSize parameter, 55

P

- page sizes, 111

- PATH_INFO, 81
- PathCheck, 42, 83
- peak concurrent users, 114
- perfdump
- about, 29-35
 - enabling, 29-30
 - sample output, 31-33
 - using to monitor server activity, 29-35
- performance
- buckets, 33
 - issues, 17-18
 - monitoring tools, 20
 - overview, 17-35
 - problems, 95
 - studies, 115-138
 - tuning, 37-94
- performance buckets
- configuration of, 34
 - defining in magnus.conf, 34
 - information in perfdump, 35
 - performance report, 34-35
 - using to monitor activity, 33
- performance monitoring, Solaris-specific, 106-107
- performance report, performance buckets, 34-35
- persistence-type, 80
- persistent connection information, 51-56
- pfx2dir function, 81
- PR_GetFileInfo, 65
- precompiled JSPs, 78
- problems
- common, 95
 - connection timeouts, 102
 - keep-alive connections flushed, 97-98
 - log file modes, 98
 - low memory, 96
 - too few threads, 96
- process modes, 42-44
- processes, 38-44
- processors, sizing issues, 113
- profiling, 22

Q

quality of service
 features, 18
 statistics and, 22
 queued connections, in JDBC resources, 74

R

ratio, hit, 61
 reapIntervalSeconds, 80
 refresh, 64
 reload-interval, 78
 restart, 64
 rlim_fd_max, 101, 108, 109

S

scalability studies, 115-138
 SE toolkit, 107
 segmap_percent, 105
 send-cgi, 33
 send-file, nocache parameter, 62
 server instances, statistics, 21
 Service, 42, 83
 servlet/JSP caching, 79
 session creation information, 56-58
 session-properties, 80
 session settings, web application, 80-81
 set-http-prop command, 56
 set-stats-prop command, 23
 single-process mode, 42-43
 SNCA, 103-104
 connection queue size, 104
 maximum threads, 104
 Solaris
 file system tuning, 101-104
 Network Cache and Accelerator, 103-104
 platform-specific issues, 101-104
 tuning for performance benchmarking, 108, 109-110
 Solaris-specific performance monitoring, 106-107
 long-term system monitoring, 107
 SE toolkit, 107

Solaris-specific performance monitoring (*Continued*)

 short-term system monitoring, 106
 sq_max_size, 103, 108, 109
 SSL performance, 19
 start options, 111
 statistics
 activating, 22-23
 connection queue, 48
 file cache information, 58
 listen socket information, 50
 monitoring, 21
 performance buckets, 33
 stats-xml
 accessing URI, 28
 enabling URI, 27
 limiting output, 28-29
 using to monitor current activity, 27-29
 studies, 115-138
 conclusion, 116
 goals, 115
 hardware used, 116
 load driver, 117
 network configuration, 118-119
 Web Server tuning, 119-120

T

table validation method, 76
 TCP buffering, tuning, 103
 tcp_conn_req_max_q, 102, 108, 109
 tcp_conn_req_max_q0, 102, 108, 109
 tcp_cwnd_max, 110
 TCP/IP, tuning, 117
 tcp_ip_abort_interval, 108, 110
 tcp_keepalive_interval, 108
 tcp_rcv_hiwat, 108, 110
 tcp_rexmit_interval_initial, 108
 tcp_rexmit_interval_max, 108
 tcp_rexmit_interval_min, 108
 tcp_slow_start_initial, 108
 tcp_smallest_anon_port, 108
 tcp_time_wait_interval, 103, 108
 tcp_xmit_hiwat, 108, 110
 tcpHalfOpenDrop, 102

- tcpListenDrop, 102
- tcpListenDropQ0, 102
- TerminateTimeout directive, 40
- test results, 115-138
- thread pools, 47, 65-68
 - custom, 40-41
 - disabled, 40
 - native thread pool, 41-42, 65-66
- threads, 38-44
 - acceptor, 51
 - creation statistics, 56-58
 - keep-alive, 55
 - maximum, 57
 - maximum and SNCA, 104
 - multi-process mode, 43
 - too few, 96
- tips, general, 37-38
- transaction isolation level, JDBC resource, 77
- tuning maxLocks, 80
- tuning MMapSessionManager, 80-81
- tuning TCP buffering, 103
- tuning the Web Server, 37-94
 - Java web applications performance, 78-81
 - threads, processes, and connections, 38-44
 - using statistics, 46-77
- tuning tips
 - general, 37-38
 - platform-specific, 101-111
- tuning Web Server
 - 6.1 to 7.0 parameter mapping, 44-46
 - keep-alive subsystem, 55

U

- UFS, 105
- under-throttled server, 96
- UNIX file system, 105
- using statistics to tune your server, 46-77

V

- validation method, JDBC resource, 76
- validation table name, JDBC resource, 76

- virtual servers
 - default, 51
 - HTTP listeners, 50
 - performance overview, 18
- vmstat 60 command, 106

W

- wait timeout, JDBC resource, 76
- web applications, 71-72
 - session timeout, 80
 - statistics for, 26
 - tuning performance, 78-81
- Web Server
 - start options, 111
 - tuning for studies, 119-120
- work queue
 - length, 66
 - limit, 66
 - peak, 66