Performance Tuning, Sizing, and Scaling Guide

Sun™ ONE Web Server

Version 6.1

817-6249-10
April 2004

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

Contents

About This Guide

This guide discusses adjustments you can make that may improve the performance of Sun™ Open Net Environment (Sun ONE) Web Server 6.1. 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, and describes the `perfdump` performance utility and tuning parameters that are built into the server.

This preface contains the following topics:

Who Should Use This Guide

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.

Using the Documentation

The Sun ONE Web Server manuals are available as online files in PDF and HTML formats from the following location:

http://docs.sun.com/db/prod/s1websrv#hic

The following table lists the tasks and concepts described in the Sun ONE Web Server manuals.

Table 1 Sun ONE Web Server Documentation Roadmap

For Information About	See the Following
Late-breaking information about the software and documentation	*Release Notes*
Getting started with Sun ONE Web Server, including hands-on exercises that introduce server basics and features (recommended for first-time users)	*Getting Started Guide*
Performing installation and migration tasks: • Installing Sun ONE Web Server and its various components, supported platforms, and environments • Migrating from Sun ONE Web Server 4.1 or 6.0 to Sun ONE Web Server 6.1	*Installation and Migration Guide*

Table 1 Sun ONE Web Server Documentation Roadmap

For Information About	See the Following
Performing the following administration tasks: • Using the Administration and command-line interfaces • 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™ 2 Platform, Enterprise Edition (J2EE™ platform) 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	*Administrator's Guide*
Using programming technologies and APIs to do the following: • Extend and modify Sun ONE Web Server • Dynamically generate content in response to client requests • Modify the content of the server	*Programmer's Guide*

Table 1 Sun ONE Web Server Documentation Roadmap

For Information About	See the Following
Creating custom Netscape Server Application Programmer's Interface (NSAPI) plugins	*NSAPI Programmer's Guide*
Implementing servlets and JavaServer Pages™ (JSP™) technology in Sun ONE Web Server	*Programmer's Guide to Web Applications*
Editing configuration files	*Administrator's Configuration File Reference Guide*
Tuning Sun ONE Web Server to optimize performance	*Performance Tuning, Sizing, and Scaling Guide*

How This Guide Is Organized

This guide is organized as follows:

- Chapter 1, "Performance and Monitoring Overview"

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

- Chapter 2, "Tuning Sun ONE Web Server"

  This chapter describes specific adjustments you can make that may improve Sun ONE Web Server performance.

- Chapter 3, "Miscellaneous Performance Topics"

  This chapter describes miscellaneous performance topics, including discussion of `magnus.conf` and `obj.conf` settings that can be used to improve server performance.

- Chapter 4, "Common Performance Problems"

  This chapter discusses common web site performance problems, and offers tips and solutions.

- Chapter 5, "Platform-specific Issues and Tips"

  This chapter provides platform-specific tuning tips.

- Chapter 6, "Sizing and Scaling Your Server"

  This chapter examines the subsystems of your server, and provides recommendations for optimal performance.

- Chapter 7, "Scalability Studies"

  This chapter describes the results of scalability studies. You can use these studies as examples of how you might configure your system to best take advantage of Sun ONE Web Server's strengths.

Documentation Conventions

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

- **File and directory paths**

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

- **URLs** are given in the format:

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

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

- **Font conventions** include:

  - The `monospace` font is used for sample code and code listings, API and language elements (such as function names and class names), file names, path names, directory names, and HTML tags.
  - *`Italic`* monospace type is used for code variables.
  - *Italic* type is also used for book titles, emphasis, variables and placeholders, and words used in the literal sense.
  - **Bold** type is used as either a paragraph lead-in or to indicate words used in the literal sense.

- **Installation root directories** are indicated by *install_dir* in this guide.

  By default, the location of *install_dir* is as follows:

  - On UNIX-based platforms: `/opt/SUNWwbsvr/`

- On Windows: `C:\Sun\WebServer6.1`

Product Support

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

- The online support web site at:

  http://www.sun.com/supportraining/

- The telephone dispatch number associated with your maintenance contract

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

- Description of the problem, including the situation where the problem occurs and its impact on your operation.
- Machine type, operating system version, and product version, including any patches and other software that might be affecting the problem.
- Detailed steps on the methods you have used to reproduce the problem.
- Any error logs or core dumps.

Chapter 1

Performance and Monitoring Overview

Sun ONE Web Server is designed to meet the needs of the most demanding, high-traffic sites in the world. It runs flexibly on UNIX, Linux, and Windows, and can serve both static and dynamically generated content. Sun ONE 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. Your environment is unique, however, so the impacts of the suggestions provided here also 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

The first step toward sizing your server is to determine your requirements. Performance means different things to users than 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 Sun ONE 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. 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.

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

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

Virtual Servers

Virtual servers add another layer to the performance improvement process. Certain settings are tunable for the entire server, 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 or class of virtual servers. For example, you can use QOS features to limit the number of connections allowed for a virtual server or class of virtual servers.

For more information about using the quality of service features, see the Sun ONE Web Server 6.1 *Administrator's Guide*.

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 Sun ONE Web Server in several different ways, as discussed in the following topics:

- Monitoring Current Activity Using the Server Manager
- Monitoring Current Activity Using the perfdump Utility
- Using Performance Buckets

See Also

General Tuning Tips
Solaris-specific Performance Monitoring

Monitoring Current Activity Using the Server Manager

You can monitor many performance statistics through the Server Manager user interface, and through `stats-xml`. Once statistics are activated, you can monitor the following areas:

- Connections
- DNS
- Keep-alive
- Cache
- Virtual Server

Activating Statistics

You must activate statistics on Sun ONE Web Server before you can monitor performance. This can be done through the Server Manager, or by editing the `obj.conf` and `magnus.conf` files.

CAUTION When you activate statistics/profiling, statistics information is made available to any user of your server.

Activating Statistics from the Server Manager

To activate statistics from the user interface:

1. From the Server Manager, click the Monitor tab, and then click Monitor Current Activity.

   The Enable Statistics/profiling page displays.

2. Select Yes to activate statistics/profiling.
3. Click OK, click Apply, and then click the Apply Changes button to activate statistics/profiling.

Activating Statistics with stats-xml

You can also activate statistics directly by editing the `obj.conf` and `magnus.conf` files. Users who create automated tools or write customized programs for monitoring and tuning may prefer to work directly with `stats-xml`.

To activate statistics using `stats-xml`:

1. Under the default object in `obj.conf`, add the following line:

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

2. Add the following `Service` function to `obj.conf`:

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

3. Add the `stats-init` SAF to `magnus.conf`.

   Here's an example of `stats-init` in `magnus.conf`:

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

The above example shows you can also designate the following:

- **update-interval.** The period in seconds between statistics updates. A higher setting (less frequent) will be better for performance. The minimum value is 1; the default value is 5.
- **virtual-servers.** The maximum number of virtual servers for which you track statistics. This number should be set equal to or higher than the number of virtual servers configured. Smaller numbers result in lower memory usage. The minimum value is 1; the default is 1000.

- **profiling.** Activate NSAPI performance profiling. The default is "no," which results in slightly better server performance. However, if you activate statistics through the user interface, profiling is turned on by default.

Monitoring Statistics

Once you've activated statistics, you can get a variety of information on how your server instance and your virtual servers are running. The statistics are broken up into functional areas.

To monitor statistics from the Server Manager:

1. From the Server Manager, click the Monitor tab, and then click Monitor Current Activity.
2. Make sure that statistics/profiling is activated ("Yes" is selected and applied for "Activate Statistics/Profiling?").
3. From the drop-down list, select a refresh interval.

   This is the interval, in seconds, that updated statistics will be displayed on your browser.
4. From the drop-down list, select the type of web server statistics to display.
5. Click Submit.

   A page appears displaying the type of statistics you selected. The page is updated every 5-15 seconds, depending on the refresh interval. All pages will display a bar graph of activity, except for Connections.
6. Select the process ID from the drop-down list.

You can view current activity through the Server Manager, but these categories are not fully relevant for tuning your server. The `perfdump` statistics are recommended for tuning your server. For more information, see "Using Statistics to Tune Your Server" on page 28.

Virtual Server Statistics

Virtual server statistics can be viewed from the Server Manager. You can choose to display statistics for the server instance, for an individual virtual server, or for all. This information is not provided through `perfdump`.

Monitoring Current Activity Using the perfdump Utility

The `perfdump` utility is a Server Application Function (SAF) built into Sun ONE Web Server that collects various pieces of performance data from the Web Server internal statistics and displays them in ASCII text. The `perfdump` utility allows you to monitor a greater variety of statistics than those available through the Server Manager.

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

Installing the perfdump Utility

To install `perfdump`, make the following modifications in `obj.conf`:

1. Add the following object to your `obj.conf` file after the default object:

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

2. Add the following to the default object:

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

   Make sure that the `.perf NameTrans` directive is specified before the `document-root NameTrans` directive in the default object.

3. If not already activated, activate `stats-xml`.

   For more information, see "Activating Statistics" on page 17.

4. Restart your server software.

5. Access `perfdump` by entering this URL:

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

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

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

See Also

"Using Statistics to Tune Your Server"

Sample perfdump Output

The following is sample `perfdump` output:

```
---------------------------------------------------------

webservd pid: 2408

ConnectionQueue:
---------------------------------
Current/Peak/Limit Queue Length     0/0/4096
Total Connections Queued            0
Average Queueing Delay              0.00 milliseconds

ListenSocket ls1:
------------------------
Address                    http://0.0.0.0:8080
Acceptor Threads           1
Default Virtual Server     https-iws-files2.red.iplanet.com

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

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

CacheInfo:
------------------
enabled             yes
CacheEntries        0/1024
Hit Ratio           0/0 (  0.00%)
Maximum Age         30

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

Server DNS cache disabled
```

```
Async DNS disabled

Performance Counters:
------------------------------------------------
                          Average        Total     Percent


Total number of requests:                    0
Request processing time:    0.0000       0.0000

default-bucket (Default bucket)
Number of Requests:                          0    (  0.00%)
Number of Invocations:                       0    (  0.00%)
Latency:                    0.0000       0.0000    (  0.00%)
Function Processing Time:   0.0000       0.0000    (  0.00%)
Total Response Time:        0.0000       0.0000    (  0.00%)

Sessions:
----------------------------
Process   Status    Function

2408      response  service-dump
------------------------------------------------------------
```

Using Performance Buckets

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

- **Name of the bucket.** This name is used for associating the bucket with a function.
- **Description.** A description of the functions that the bucket is associated with.
- **Number of requests for this function.** The total number of requests that caused this function to be called.

- **Number of times the function was invoked.** This number may not coincide with the number of requests for the function because some functions may be executed more than once for a single request.
- **Function latency or the dispatch time.** The time taken by the server to invoke the function.
- **Function time.** The time spent in the function itself.

The `default-bucket` is 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 measurement as described in "Monitoring Current Activity Using the perfdump Utility" on page 20.

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 example above creates 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 you wish 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>
```

Performance Report

The server statistics in buckets can be accessed using the `perfdump` utility. 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 differs from the number of requests in that a function could 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 Sun ONE Web Server takes to prepare for calling the function.
- `Function Processing Time` is the time in seconds Sun ONE Web Server spent inside the function. The percentage of `Function Processing Time` and `Total Response Time` is calculated with reference to the total Request `Processing Time`.

- `Total Response Time` is the sum in seconds of `Function Processing Time` and `Latency`.

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

```
Performance Counters:
-----------------------------------------------
                           Average        Total     Percent


Total number of requests:                    0
Request processing time:    0.0000       0.0000

default-bucket (Default bucket)
Number of Requests:                          0    (  0.00%)
Number of Invocations:                       0    (  0.00%)
Latency:                    0.0000       0.0000    (  0.00%)
Function Processing Time:   0.0000       0.0000    (  0.00%)
Total Response Time:        0.0000       0.0000    (  0.00%)
```

Chapter 2

Tuning Sun ONE Web Server

This chapter describes specific adjustments you can make that may improve Sun ONE Web Server performance. The chapter includes the following topics:

CAUTION 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 stepwise changes in succession, rather than trying to make a drastic change all at once. Different systems face different circumstances, and you may leap right past 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, it is strongly recommended that you start over with an unmodified `/etc/system` file.

- **Stay Informed**

  Read the Sun ONE Web Server and Solaris release notes whenever you upgrade your system. The release notes often provide updated information about specific adjustments.

Using Statistics to Tune Your Server

This section describes the information available through the `perfdump` utility, and discusses how to tune some parameters to improve your server's performance.

The default tuning parameters are appropriate for all sites except those with very high volume. The only parameters that large sites may regularly need to change are `RqThrottle`, `MaxKeepAliveConnections`, and `KeepAliveTimeout`, which are tunable from `magnus.conf` and the Server Manager.

The `perfdump` utility monitors statistics in the following categories, which are described in this section:

NOTE For general information about `perfdump`, see "Monitoring Current Activity Using the perfdump Utility" on page 20.

Once you have viewed the statistics you need, you can tune various aspects of your server's performance using:

- The `magnus.conf` file
- The Server Manager Preferences tab

The Server Manager Preferences tab includes many interfaces for setting values for server performance, including the Performance Tuning page and the File Cache Configuration page.

The Magnus Editor allows you to set values for numerous directives in the following categories, which are accessible from the drop-down list:

- DNS Settings
- SSL Settings
- Performance Settings
- CGI Settings
- Keep-Alive Settings
- Logging Settings
- Language Settings

Connection Queue Information

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

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

```
ConnectionQueue:
---------------------------------
Current/Peak/Limit Queue Length     0/0/4096
Total Connections Queued            0
Average Queueing Delay              0.00 milliseconds
```

Current /Peak /Limit

`Current/Peak/Limit queue length` shows, in order:

- The number of connections currently in the queue
- The largest number of connections that have been in the queue simultaneously
- The maximum size of the connection queue

Tuning

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

You can increase the connection queue size by:

- Setting or changing the value of `ConnQueueSize` in the Magnus Editor of the Server Manager
- Editing the `ConnQueueSize` directive in `magnus.conf`

Total Connections Queued

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

This setting is not tunable.

Average Queuing Delay

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

This setting is not tunable.

Listen Socket Information

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

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

```
ListenSocket ls1:
-----------------------
Address                    http://0.0.0.0:8080
Acceptor Threads           1
Default Virtual Server     https-iws-files2.red.iplanet.com
```

Tuning

You can create listen sockets through the Server Manager, and edit much of a listen socket's information. For more information about adding and editing listen sockets, see the Sun ONE Web Server 6.1 *Administrator's Guide*.

If you have created multiple listen sockets, `perfdump` displays all of them.

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

- Editing the `ListenQ` parameter in `magnus.conf`
- Setting or changing the `ListenQ` value in the Magnus Editor of the Server Manager
- Entering the value in the Listen Queue Size field of the Performance Tuning page of the Server Manager

Address

The `Address` field contains the base address that this listen socket is listening on. It contains the IP address and the port number.

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

Tuning

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

For more information about adding and editing listen sockets, see the Sun ONE Web Server 6.1 *Administrator's Guide*.

Acceptor Threads

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

Tuning

You can tune this number through the user interface when you edit a listen socket.

For more information about adding and editing listen sockets, see the Sun ONE Web Server 6.1 *Administrator's Guide*.

Default Virtual Server

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

Tuning

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

You can specify a default virtual server for a listen socket by:

- Setting or changing the default virtual server information using the Edit Listen Sockets page on the Preferences tab of the Server Manger.
- Editing the `defaultvs` attribute of the `CONNECTIONGROUP` element in the `server.xml` file. For more information about `server.xml`, see the Sun ONE Web Server 6.1 *Administrator's Configuration File Reference*.

Keep-Alive/Persistent Connection Information

This section provides information about the server's HTTP-level keep-alive system. For additional tuning information, see "Keep-Alive Subsystem Tuning" on page 51.

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

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

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

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

To deal with this problem, the server maintains a "Maximum number of waiting keep-alive connections" counter. A "waiting" keep-alive connection has fully completed processing the previous request, and is 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 ONE 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:

- `KeepAliveTimeout` is set to 0.
- `MaxKeepAliveConnections` count is exceeded.
- Dynamic content, such as a CGI, does not have an HTTP `content-length` header set. This applies only to HTTP/1.0 requests. If the request is HTTP/1.1, the server honors keep-alive requests even if the `content-length` is not set. The server can use chunked encoding for these requests if the client can handle them (indicated by the request header `transfer-encoding: chunked`). For more information about chunked encoding, see the Sun ONE Web Server 6.1 *NSAPI Programmer's Guide.*
- Request is not `HTTP GET` or `HEAD`.
- The request was determined to be bad. For example, if the client sends only headers with no content.

KeepAliveThreads

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

- Editing the `KeepAliveThreads` parameter in `magnus.conf`
- Setting or changing the `KeepAliveThreads` value in the Magnus Editor of the Server Manager

KeepAliveCount

This setting has two numbers:

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

Tuning

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

- Editing the `MaxKeepAliveConnections` parameter in the `magnus.conf` file
- Setting or changing the `MaxKeepAliveConnections` value in the Magnus Editor of the Server Manager

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

KeepAliveHits

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

This setting is not tunable.

KeepAliveFlushes

The number of times the server had to close a connection because the `KeepAliveCount` exceeded the `MaxKeepAliveConnections`. In the current version of the server, the server does not close existing connections when the `KeepAliveCount` exceeds the `MaxKeepAliveConnections`. Instead, new keep-alive connections are refused and the `KeepAliveResusals` count is incremented.

KeepAliveRefusals

The number of times the server could not hand off the connection to a keep-alive thread, possibly due to too many persistent connections (or when `KeepAliveCount` exceeds `MaxKeepAliveConnections`). Suggested tuning would be to increase `MaxKeepAliveConnections`.

KeepAliveTimeout

The time (in seconds) before idle keep-alive connections are closed.

KeepAliveTimeouts

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

UseNativePoll

This option is not displayed in `perfdump` or Server Manager statistics. However, for UNIX/Linux users, it should be enabled for maximum performance.

To enable native poll for your keep-alive system from the Server Manager, follow these steps:

1. Go to the Server Manager Preferences tab and select the Mangus Editor.
2. From the drop-down list, choose Keep-Alive Settings and click Manage.
3. Use the drop-down list to set `UseNativePoll` to On.
4. Click OK, and then click Apply.
5. Select Apply Changes to restart the server for your changes to take effect.

Session Creation Information

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

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

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

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

Reaching the maximum number of configured threads is not necessarily undesirable, 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 will queue up in the connection queue, potentially overflowing it. If you check your `perfdump` output on a regular basis and notice that total sessions created is often near the `RqThrottle` maximum, you should consider increasing your thread limits.

Tuning

You can increase your thread limits by:

- Editing the `RqThrottle` parameter in `magnus.conf`
- Setting or changing the `RqThrottle` value in the Magnus Editor of the Server Manager
- Entering the value in the Maximum Simultaneous Requests field of the Performance Tuning page in the Server Manager

Cache Information

The cache information section provides statistics on how your file cache is being used. The file cache caches static content so that the server handles requests for static content quickly. For tuning information, see "Tuning the File Cache" on page 55.

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

```
CacheInfo:
------------------
enabled             yes
CacheEntries        0/1024
Hit Ratio           0/0 (  0.00%)
Maximum Age         30
```

enabled

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

Tuning

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

- Unselecting it from the File Cache Configuration page under Preferences in the Server Manger.
- Editing the `FileCacheEnable` parameter in the `nsfc.conf` file. For more information about this file, see the Sun ONE Web Server 6.1 *Administrator's Configuration File Reference*.

CacheEntries

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

Tuning

You can set the maximum number of cached entries by:

- Entering a value in the Maximum # of Files field on the File Cache Configuration page under Preferences in the Server Manger
- Creating or editing the `MaxFiles` parameter in the `nsfc.conf` file. For more information about this file, see the Sun ONE Web Server 6.1 *Administrator's Configuration File Reference*.

Hit Ratio (CacheHits / CacheLookups)

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

This setting is not tunable.

Maximum Age

This 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

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

- Entering or changing the value in the Maximum Age field of the File Cache Configuration page in the Server Manager.
- Editing the `MaxAge` parameter in the `nsfc.conf` file. For more information about this file, see the Sun ONE Web Server 6.1 *Administrator's Configuration File Reference*.

Thread Pools

Three types of thread pools can be configured through the Server Manager:

- Thread Pools (UNIX/Linux)
- Native Thread Pools (Windows)
- Generic Thread Pools (Windows)

Thread Pools (UNIX/Linux Only)

Since threads on UNIX/Linux are always operating system (OS)-scheduled, as opposed to user-scheduled, UNIX/Linux users do not need to use native thread pools, and this option is not offered in the user interface for these platforms. However, you can edit the OS-scheduled thread pools and add new thread pools if needed, using the Server Manager.

Native Thread Pools (Windows Only)

On Windows, the native thread pool (`NativePool`) is used internally by the server to execute NSAPI functions that require a native thread for execution. Windows users can edit native thread pool settings using the Server Manager.

Sun ONE Web Server uses 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. This usually means it is a native OS thread. 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 plugins such as `NameTrans`, `Service`, or `PathCheck` functions, these execute by default on a thread from the native thread pool. If your plugin 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.

To do this, add the following to the "`load-modules`" `Init` line in the `magnus.conf` file:

```
Init funcs="pcheck_uri_clean_fixed_init"
shlib="C:/Netscape/p186244/P186244.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.

Generic Thread Pools (Windows Only)

On Windows, you can set up additional thread pools using the Server Manger. Use thread pools to put a limit on the maximum number of requests answered by a service function at any moment. Additional thread pools are a way to run thread-unsafe plugins. By defining a pool with a maximum number of threads set to 1, only one request is allowed into the specified service function.

Idle /Peak /Limit

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

Tuning

You can modify the `NativePoolMaxThreads` by:

- Editing the `NativePoolMaxThreads` parameter in `magnus.conf`
- Entering or changing the value in the Maximum Threads field of the Native Thread Pool page in the Server Manager

Work Queue Length /Peak /Limit

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

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

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

Tuning

You can modify the `NativePoolQueueSize` by:

- Editing the `NativePoolQueueSize` parameter in `magnus.conf`
- Entering or changing the value in the Queue Size field of the Native Thread Pool page in the Server Manager

NativePoolStackSize

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` parameter in `magnus.conf`
- Setting or changing the `NativePoolStackSize` value in the Magnus Editor of the Server Manager
- Entering or changing the value in the Stack Size field of the Native Thread Pool page in the Server Manager

NativePoolQueueSize

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 `RqThrottle` 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 if `LogVerbose` is enabled. Setting the `NativePoolQueueSize` higher than `RqThrottle` 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, letting 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 `RqThrottle` 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` parameter in `magnus.conf`
- Entering or changing the value in the Queue Size field of the Native Thread Pool page in the Server Manager

NativePoolMaxThreads

`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 will not need to increase this number, but if you are not saturating your CPU and you are seeing requests queue up, then you should increase this number.

Tuning

You can modify the `NativePoolMaxThreads` by:

- Editing the `NativePoolMaxThreads` parameter in `magnus.conf`
- Entering or changing the value in the Maximum Threads field of the Native Thread Pool page in the Server Manager

NativePoolMinThreads

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`
- Setting or changing the `NativePoolMinThreads` value in the Magnus Editor of the Server Manager
- Entering or changing the value in the Minimum Threads field of the Native Thread Pool page in the Server Manager

DNS Cache Information

The DNS cache caches IP addresses and DNS names. Your server's DNS cache is disabled by default. Statistics are displayed in the DNS Statistics for Process ID page under Monitor in the Server Manager.

enabled

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

Tuning

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

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

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

- Setting the DNS value to "on" in the Magnus Editor of the Server Manager
- Selecting DNS Enabled from the Performance Tuning page under Preferences in the Server Manger

CacheEntries (CurrentCacheEntries / MaxCacheEntries)

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

Tuning

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

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

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

  The default cache size is 1024

- Entering or changing the value in the Size of DNS cache field of the Performance Tuning page in the Server Manager

HitRatio (CacheHits / CacheLookups)

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

This setting is not tunable.

Busy Functions

The default busy function returns a "503 Service Unavailable" response and logs a message if `LogVerbose` is enabled. You may wish to modify this behavior for your application. 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 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 your busy function will apply to all functions that require a native thread to execute when the default thread type is non-native.

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

```
extern "C" NSAPI_PUBLIC int my_custom_busy_function(pblock *pb,
Session *sn, Request *rq);
my_init(pblock *pb, Session *, Request *)
{
func_insert("service-toobusy", my_custom_busy_function);
}
```

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

Threads, Processes, and Connections

This section includes the following topics:

Connection-Handling Overview

In Sun ONE Web Server, acceptor threads on a listen socket accept connections and put them into a connection queue. Session threads then pick up connections from the queue and service the requests. The session threads post more session threads if required at the end of the request. The policy for adding new threads is based on the connection queue state:

- Each time a new connection is returned, the number of connections waiting in the queue (the backlog of connections) is compared to the number of session threads already created. If it is greater than the number of threads, more threads are scheduled to be added the next time a request completes.
- The previous backlog is tracked, so that if it is seen to be increasing over time, and if the increase is greater than the `ThreadIncrement` value, and the number of session threads minus the backlog is less than the `ThreadIncrement` value, then another `ThreadIncrement` number of threads are scheduled to be added.
- The process of adding new session threads is strictly limited by the `RqThrottle` value.
- To avoid creating too many threads when the backlog increases suddenly (such as the startup of benchmark loads), the decision as to whether more threads are needed is made only once every 16 or 32 times a connection is made based on how many session threads already exist.

The following directives that affect the number and timeout of threads, processes, and connections can be tuned in the Magnus Editor or `magnus.conf`:

- `AcceptTimeout`
- `ConnQueueSize`
- `HeaderBufferSize`

- KeepAliveThreads
- KeepAliveTimeout
- KernelThreads
- ListenQ
- MaxKeepAliveConnections
- MaxProcs (UNIX Only)
- PostThreadsEarly
- RcvBufSize
- RqThrottle
- RqThrottleMin
- SndBufSize
- StackSize
- StrictHttpHeaders
- TerminateTimeout
- ThreadIncrement
- UseNativePoll (UNIX only)

For detailed information about these directives, see the Sun ONE Web Server 6.1 *Administrator's Configuration File Reference*.

Process Modes

You can run Sun ONE Web Server in one of the following two modes:

- Single-Process Mode
- Multi-Process Mode

Single-Process Mode

In the single-process mode the server receives requests from web clients to a single process. Inside the single server process many threads are running that are waiting for new requests to arrive. When a request arrives, it is handled by the thread receiving 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 plugins provided by Netscape/Sun ONE are thread-safe and thread-aware, providing good scalability and concurrency. However, your legacy applications may 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/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 ONE Web Server. However, because all of the Netscape/Sun ONE extensions are built to support a single-process threaded environment, they may not run in the multi-process mode, and the Search plugins will fail on startup if the server is in multi-process mode.

In the multi-process mode, the server spawns multiple server processes at startup. Each process contains one or more threads (depending on the configuration) 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 48 for information about the `MaxProcs` directive). However, many modern operating systems will not distribute connections evenly, particularly when there are a small number of concurrent connections.

Because Sun ONE Web Server cannot guarantee that load is distributed evenly among server processes, you may encounter performance problems if you specify `RqThrottle` 1 and `MaxProcs` greater than 1 to accommodate a legacy application that is not thread-safe. The problem will be especially pronounced if the legacy application takes a long time to respond to requests (for example, if the legacy application contacts a backend database). In this scenario, it may be preferable to

use the default value for `RqThrottle` and serialize access to the legacy application using thread pools. For more information about creating a thread pool, refer to the description of the `thread-pool-init` SAF in the Sun ONE Web Server 6.1 *NSAPI Programmer's Guide*.

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)

Use this directive to set your UNIX/Linux server in multi-process mode, which may allow for higher scalability on multi-processor machines. If you set the value to less than 1, it will be ignored and the default value of 1 will be used. See "Multi-Process Mode" on page 47 for a discussion of the performance implications of setting this to a value greater than 1.

Tuning

You can set the value for `MaxProcs` by:

- Editing the `MaxProcs` parameter in `magnus.conf`
- Setting or changing the MaxProcs value in the Magnus Editor of the Server Manager

NOTE You will receive duplicate startup messages when running your server in `MaxProcs` mode.

Listen Socket Acceptor Threads

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

Tuning

You can set the number of listen socket acceptor threads by:

- Editing the `server.xml` file
- Entering the number of acceptor threads you want in the Number of Acceptor Threads field of the Edit Listen Socket page of the Server Manager

Maximum Simultaneous Requests

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

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

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

However, in actuality, Internet clients frequently connect to the server and then do not complete their requests. In these cases, the server waits 30 seconds or more for the data before timing out. You can define this timeout period using the `AcceptTimeout` directive in `magnus.conf`. The default value is 30 seconds. By setting it to less than the default 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 may need to increase the number of maximum simultaneous requests. For more information about `AcceptTimeout`, see the Sun ONE Web Server 6.1 *Administrator's Configuration File Reference*.

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

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

NOTE If you are using older NSAPI plugins that are not reentrant, they will not work with the multi-threading model described in this document. To continue using them, you should revise them so that they are reentrant. If this is not possible, you can configure your server to work with them by setting `RqThrottle` to 1, and then using a high value for `MaxProcs`, such as 48 or greater, but this will adversely impact your server's performance.

NOTE When configuring Sun ONE Web Server to be used with SNCA (the Solaris Network Cache and Accelerator), setting the `RqThrottle` and `ConnQueueSize` parameters 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 `RqThrottle` and `ConnQueueSize` must *both* be set to 0.

For more information about `RqThrottle` and `ConnQueueSize`, see the chapter pertaining to `magnus.conf` in the Sun ONE Web Server 6.1 *Administrator's Configuration File Reference*. Also consult the `RqThrottle` and `ConnQueueSize` entries in the index in this book. For information about using SNCA, see "Using the Solaris Network Cache and Accelerator (SNCA)" on page 88.

Tuning

You can tune the number of simultaneous requests by:

- Editing `RqThrottleMin` and `RqThrottle` in the `magnus.conf` file
- Entering or changing values for the `RqThrottleMin` and `RqThrottle` fields in the Magnus Editor of the Server Manager
- Entering the desired value in the Maximum Simultaneous Requests field from the Performance Tuning page under Preferences in the Server Manger

Keep-Alive Subsystem Tuning

The keep-alive (or HTTP/1.1 persistent connection handling) subsystem in Sun ONE Web Server 6.1 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.

There are several tuning parameters that can help improve performance. Those parameters are listed below:

- `acceptorthreads`: Number of threads waiting to accept incoming connections on a given network port. This is specified per the listen socket (`LS`) element in `server.xml`.

- `ConnQueueSize`: Size of the queue of active, ready-to-process connections.
- `RqThrottle`: Number of worker threads in the server. Each thread parses and services a request from an active connection. Worker threads, in contrast with acceptor threads, service requests. The maximum number of worker threads is configured using `RqThrottle`. For more information, see "Maximum Simultaneous Requests" on page 49.
- `MaxKeepAliveConnections`: This controls the maximum number of keep-alive connections the Web Server can maintain at any time. The default is 256. The range is 0 to 32768.
- `KeepAliveTimeout`: This directive determines the maximum time (in seconds) that the server holds open an HTTP keep-alive connection or a persistent connection between the client and the server. The default is 30 seconds. The connection will timeout if idle for more than 30 seconds. The maximum is 300 seconds (5 minutes).
- `KeepAliveThreads`: This directive determines the number of threads in the keep-alive subsystem. It is recommended that this number be a small multiple of the number of processors on the system (for example, a 2 CPU system should have 2 or 4 keep-alive threads). The default is 1.
- `KeepAliveQueryMaxSleepTime`: Specifies an upper limit to the time slept (in milliseconds) after polling keep-alive connections for further requests. The default is 100. On lightly loaded systems that primarily service keep-alive connections, you can lower this number to enhance performance. Doing so can increase CPU usage, however.
- `KeepAliveQueryMeanTime`: Specifies the desired keep-alive latency in milliseconds. The default value of 100 is appropriate for almost all installations. Note that CPU usage will increase with lower `KeepAliveQueryMeanTime` values.

For more information about the Web Server's keep-alive subsystem, see "Keep-Alive/Persistent Connection Information" on page 33.

For information about connection queue sizing, see "Connection Queue Information" on page 29.

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 may want to set it to 2.

Example

In the following example, acceptor threads are increased, and keep-alive connections are reduced:

In magnus.conf:

```
MaxKeepAliveConnections 0
RqThrottle 128
RcvBufSize 8192
```

In server.xml:

```
<SERVER legacyls="ls1">
    <LS id="ls1" ip="0.0.0.0" port="8080" security="off"
blocking="no"
        acceptorthreads="2"
</SERVER>
```

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

If users are experiencing connection timeouts from a browser to Sun ONE Web Server when the server is heavily loaded, you can increase the size of the HTTP listener backlog queue by setting the `ListenQ` parameter in the `magnus.conf` file to:

```
ListenQ  8192
```

The `ListenQ` parameter specifies the maximum number of pending connections on a listen socket. Connections that time out on a listen socket whose backlog queue is full will fail.

HTTP/1.1-style Workload

In general, it is a tradeoff between throughput and latency while tuning server persistent connection handling. The `KeepAliveQueryQuery*` directives (`KeepAliveQueryMeanTime` and `KeepAliveQueryMaxSleepTime`) control latency. Lowering the values of these directives is intended to lower latency on lightly loaded systems (for example, reduce page load times). Increasing the values of these directives is intended to raise aggregate throughput on heavily loaded systems (for example, increase the number of requests per second the server can handle). However, if there's too much latency and too few clients, aggregate throughput will suffer 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 `KeepAliveQueryMeanTime` and/or `KeepAliveQueryMaxSleepTime`.
- If there's no idle CPU time, increase `KeepAliveQueryMeanTime` and/or `KeepAliveQueryMaxSleepTime`.

For more information about these directives, see "Keep-Alive Subsystem Tuning" on page 51.

Also, chunked encoding could affect the performance for HTTP/1.1 workload. Tuning the response buffer size could positively affect the performance. A higher `OutputStreamSize` for a plugin would result in sending `Content-length:` header, instead of chunking the response.

Example

In the following example, `MaxKeepAliveConnections` is increased, as is `UseOutputStreamSize` for the `nsapi_test Service` function:

In magnus.conf:

```
MaxKeepAliveConnections 8192
KeepAliveThreads 2
UseNativePoll 1
RqThrottle 128
RcvBufSize 8192
```

In obj.conf:

```
<Object name="nsapitest">
ObjectType fn="force-type" type="magnus-internal/nsapitest"
Service method=(GET) type="magnus-internal/nsapitest"
fn="nsapi_test"
UseOutputStreamSize=8192
</Object>
```

Tuning the File Cache

Sun ONE Web Server uses a file cache to serve static information faster. In previous versions of the server, there was also an accelerator cache that routed requests to the file cache, but the accelerator cache is no longer used. 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.

This section includes the following topics:

Configuring the File Cache

The file cache is turned on by default. The file cache settings are contained in a file called `nsfc.conf`. You can use the Server Manager to change the file cache settings. For more information about nsfc.conf, see the Sun ONE Web Server 6.1 *Administrator's Configuration File Reference*.

To configure the file cache, follow these steps:

1. From the Server Manager, select the Preferences tab.
2. Select File Cache Configuration.
3. Select Enable File Cache, if not already selected.
4. Choose whether to transmit files.

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

   The default size is twice the maximum number of files plus 1. For example, if your maximum number of files is set to 1024, the default hash table size is 2049.
6. Enter a maximum age in seconds for a valid cache entry.

   By default, this is set to 30.

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

   Set the maximum age based on whether the content is updated (existing files are modified) on a regular schedule. For example, if content is updated four times a day at regular intervals, you could set the maximum age to 21600 seconds (6 hours). Otherwise, consider setting the maximum age to the longest time you are willing to serve the previous version of a content file after the file has been modified.

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

   By default, this is set to 1024.

8. (UNIX/Linux only) Enter medium and small file size limits in bytes.

   By default, the Medium File Size Limit is set to 537600.

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

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

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

9. (UNIX/Linux only) Set the medium and small file space.

   The medium file space is the size in bytes of the virtual memory used to map all medium sized files. By default, this is set to 10485760.

   The small file space is the size of heap space in bytes used for the cache, including heap space used to cache small files. By default, this is set to 1048576 for UNIX/Linux.

10. Click OK, and then click Apply.

11. Select Apply Changes to restart your server and put your changes into effect.

Using the nocache Parameter

You can use the parameter `nocache` for the `Service` function `send-file` to specify that files in a certain directory should not be cached. For example, if you have a set of files that changes too rapidly for caching to be useful, you can put them 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 above example, the server does not cache static files from `/export/mydir/` when requested by the URL prefix `/myurl`.

Monitoring the File Cache with the Server Manager

To view the file cache statistics with the Server Manager, do the following:

1. From the Server Manager, select Monitor.
2. Select Monitor Current Activity.

   If you have not yet activated statistics, do so when the Enable Statistics/Profiling page displays, click OK, and then restart the server and return to this page.
3. Select a refresh interval from the drop-down list.

4. From the drop-down list of statistics to be displayed, choose Cache and then click Submit.
5. The cache statistics display and are refreshed every 5-15 seconds, depending on the refresh interval.

The statistics include information on your cache settings, how many hits the cache is getting, and so on.

File Cache Dynamic Control and Monitoring

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

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

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

```
Sun ONE Web Server File Cache Status (pid 7960)

The file cache is enabled.

Cache resource utilization

Number of cached file entries = 1039 (112 bytes each, 116368 total
bytes)
Heap space used for cache = 237641/1204228 bytes
Mapped memory used for medium file contents = 5742797/10485760
bytes
Number of cache lookup hits = 435877/720427 ( 60.50 %)
Number of hits/misses on cached file info = 212125/128556
Number of hits/misses on cached file content = 19426/502284
Number of outdated cache entries deleted = 0
Number of cache entry replacements = 127405
Total number of cache entries deleted = 127407
Number of busy deleted cache entries = 17

Parameter settings

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

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

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

- `?stop`: Shuts down the cache.

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

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

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

Tuning the ACL User Cache

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

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

There can also be a potential (but much harder to hit) bottleneck with the number of groups stored in a cache entry (4 by default). If a user belongs to 5 groups and hits 5 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 2 cache entries, the entry with the original group information is ignored.

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

This section includes the following topics:

ACL User Cache Directives

To adjust the ACL user cache values you must manually add the following directives to your `magnus.conf` file:

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

ACLCacheLifetime

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

ACLUserCacheSize

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

ACLGroupCacheSize

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

Verifying ACL User Cache Settings

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

```
User authentication cache entries expire in ### seconds.

User authentication cache holds ### users.

Up to ### groups are cached for each cached user.
```

Tuning

You can turn `LogVerbose` on by:

- Editing the `LogVerbose` parameter in `magnus.conf`
- Setting or changing the `LogVerbose` value to "on" in the Magnus Editor of the Server Manager

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

Tuning Java Web Applications Performance

This section includes the following topics:

- Configuring the Web Application's Session Settings
- Configuring JDBC Connection Pooling
- JDBC Connection Pool Attributes

Using Java Heap Tuning

As with all Java programs, the performance of the web applications in the Sun ONE Web Server is dependent on the heap management performed by the virtual machine (VM). There is a trade-off between pause times and throughput. A good place to start is by reading the performance documentation for the Java HotSpot virtual machine, which can be found at the following location:

`http://java.sun.com/docs/hotspot/index.html`

Java VM options are specified using the `JVMOPTIONS` subelement of the `JAVA` element in `server.xml`. For more information, see the Sun ONE Web Server 6.1 *Administrator's Configuration File Reference*.

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 precompiling your JSPs before deploying them onto your server. For more information about `jsp-config` and about precompiling JSPs for Sun ONE Web Server, see the Sun ONE Web Server 6.1 *Programmer's Guide to Web Applications*. Also see the following section, "Configuring Class Reloading."

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. For example, this is useful for common queries that all visitors to your site run: you want the results of the query to be dynamic because it might change day to day, 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 information about caching servlet results in the Sun ONE Web Server 6.1 *Programmer's Guide to Web Applications*.

Configuring the Java Security Manager

Sun ONE Web Server 6.1 supports the Java Security Manager. The main drawback of running with the Security Manager is that it negatively impacts performance. The Java Security Manager is disabled by default when you install the product. Running without the Security Manager may improve performance significantly for some types of applications. Based on your application and deployment needs, you should evaluate whether to run with or without the Security Manager. For more information, see the Sun ONE Web Server 6.1 *Programmer's Guide to Web Applications*.

Configuring Class Reloading

The `dynamicreloadinterval` of the `JAVA` element in `server.xml` and the `dynamic-reload-interval` of the `class-loader` element in `sun-web.xml` controls 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. In a production environment where changes are made in a scheduled manner, set this value to -1 to prevent the server from constantly checking for updates. The default value is -1 (that is, class reloading is disabled). For more information about elements in `server.xml`, see the Sun ONE Web Server 6.1 *Administrator's Configuration File Reference*. For more information about elements in `sun-web.xml`, see the Sun ONE Web Server 6.1 *Programmer's Guide to Web Applications*. Also see the previous section in this guide, "Using Precompiled JSPs."

Avoiding Directories in the Classpath

For certain applications (especially if the Java Security Manager is enabled) you can improve performance by ensuring that there are no directories in the classpath. To do so, ensure that there are no directories in the `classpath` elements in `server.xml` (`serverclasspath`, `classpathprefix`, `classpathsuffix`). For more information about these elements, see the Sun ONE Web Server 6.1 *Administrator's Configuration File Reference*. 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-lived 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-lived 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 the Sun ONE Web Server 6.1 *Programmer's Guide to Web Applications*.

In multi-process mode when the `persistence-type` in `sun-web.xml` is configured to be either `s1ws60` or `mmap`, the session manager uses cross-process locks to ensure session data integrity. These can be configured to improve performance as described below.

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) will contend for the same lock. Increasing `maxLocks` will reduce the number of sessions that contend for the same lock and may 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 the "Session Managers" chapter in the Sun ONE Web Server 6.1 *Programmer's Guide to 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 (documented for the MMapSessionManager in the Sun ONE Web Server 6.1 *Programmer's Guide to Web Applications*):

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

This example would create 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 will be the increase in process size.

Configuring JDBC Connection Pooling

A JDBC connection pool is a named group of JDBC connections to a database. These connections are created when the first request for connection is made on the pool when you start Sun ONE Web Server.

The JDBC connection pool defines the properties used to create a connection pool. Each connection pool uses a JDBC driver to establish a connection to a physical database at server start-up.

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 will be using 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 connection pools can be created and edited using the Administration interface, or by editing the attributes of the `JDBCCONNECTIONPOOL` element in the `server.xml` file. For more information, see the Sun ONE Web Server 6.1 *Administrator's Guide* and the Sun ONE Web Server *6.1 Administrator's Configuration File Reference*, respectively.

NOTE Each defined pool is instantiated during web server startup. However, the connections are only created the first time the pool is accessed. It is recommended that you jump-start a pool before putting it under heavy load.

JDBC Connection Pool Attributes

Depending on your application's database activity, you may need to size connection pool attributes. Attributes of a JDBC connection pool are listed below, along with considerations relating to performance.

name

The pool name.

datasourceclassname

The jdbc driver class that implements `javax.sql.DataSource`.

steadypoolsize

The size the pool will tend to keep during the life of the server instance. Also 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. This will minimize creation of connections during the life of the application, and will minimize pool resizing. Use a lower number if the pool load is expected to be small. This will minimize resource consumption.

maxpoolsize

The maximum number of connections that a pool can have at any given time. Defaults to 32.

Use this parameter 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.

poolresizequantity

Number of connections to be removed when the `idletimeout` timer expires. Connections that have been idle longer than the timeout are candidates for removal. When the pool size reaches steady-pool-size, the connection removal stops. Defaults to 2.

Keep this number low for pools that expect regular and steady changes in demand. A higher number is recommended for pools that expect infrequent and pronounced changes in the load.

idletimeout

The maximum amount in seconds that a connection is ensured to remain unused in the pool. Also the intervals at which the resizer task will be scheduled.

Note that this does not control connection timeouts enforced at the database server side. Defaults to 300.

Setting this attribute to 0 prevents the connections from being closed and causes the resizing task not to be scheduled. This is recommended for pools that expect continuous high demand. Otherwise, administrators are advised to keep this timeout shorter than the database server-side timeout (if such timeouts are configured on the specific vendor's database), to prevent accumulation of unusable connections in the pool.

maxwaittime

The amount of time in milliseconds that a request waits for a connection in the queue before timing out. Defaults to 60000.

Setting this attribute to 0 causes a request for a connection to wait indefinitely. This could also improve performance by keeping the pool from having to account for connection timers.

connectionvalidationrequired

If set to `true`, the pool will always execute a call on the connection to verify its validity. Defaults to `off`.

The overhead caused by this call can be avoided by setting the parameter to `false`.

connectionvalidationmethod

The method used for validation. Defaults to `auto-commit`.

If validation is needed, the methods `auto-commit` and `meta-data` are less costly than the method `table`. The first two require a method call, but they might not be effective if the JDBC driver caches the result of the call. The third method is almost always effective, but it requires the execution of a SQL statement, and thus is less performance-friendly.

validationtablename

The user-defined table to be use for validation. Defaults to `test`.

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

failallconnections

Indicates whether all connection in the pool are re-created when one is found to be invalid or only the invalid one. Only applicable if `connectionvalidationrequired` is set to `true`. Defaults to `off`.

If set to `true`, all of the re-creation work will be done in one step, and the thread requesting the connection will be heavily affected. If set to `false`, the load of re-creating connections will be distributed between the threads requesting each connection.

transactionisolationlevel

Specifies the Transaction Isolation Level on the pooled database connections. This setting is optional and has no default.

If left empty, the default isolation level of the connection will be left intact. Setting it to any value will incur the small performance penalty cause by the method call.

isolationlevelguaranteed

Only applicable if a `transactionisolationlevel` has been specified. Defaults to `off`.

Leaving this as `off` or `false` will cause the isolation level to be set only when the connection is created. Setting this to `true` will set the level every time the connection is leased to an application. It is recommended that you leave this set to `false`.

Chapter 3

Miscellaneous Performance Topics

This chapter provides miscellaneous performance information and includes the following topics:

- Miscellaneous magnus.conf Directives
- Miscellaneous obj.conf Parameters
- Using Quality of Service
- Using Load Balancing

Miscellaneous magnus.conf Directives

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

- Buffer Size
- Connection Timeout
- CGIStub Processes (UNIX/Linux)

For a complete list and description of `magnus.conf` directives, see the Sun ONE Web Server 6.1 *Administrator's Configuration File Reference*.

Buffer Size

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

Tuning

You can set the buffer size by:

- Editing the `SndBufSize` and `RcvBufSize` parameters in `magnus.conf`
- Setting or changing the `SndBufSize` and `RcvBufSize` values in the Magnus Editor of the Server Manager

Connection Timeout

You can specify the number of seconds the server waits for data to arrive from the client before closing the connection by using the `AcceptTimeout` directive. If data does not arrive before the timeout expires, the connection is closed. This is set to 30 seconds by default. Under most circumstances, you won't need to change this setting. You can free up threads by setting this to less than the default, but you might also disconnect users with slower connections.

Tuning

You can set `AcceptTimeout` by:

- Editing the `AcceptTimeout` parameter in `magnus.conf`
- Setting or changing the `AcceptTimeout` value in the Magnus Editor of the Server Manager

CGIStub Processes (UNIX/Linux)

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

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

The four directives and their defaults that can be tuned to control `Cgistub` are:

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

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

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

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

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

Tuning

You can set the `CGIStub` processes by:

- Editing them in `magnus.conf`
- Setting or changing their values in the Magnus Editor of the Server Manager

Miscellaneous obj.conf Parameters

You can use some `obj.conf` function parameters to improve your server's performance, as discussed in the topics in this section:

- find-pathinfo-forward
- nostat

In addition to these parameters, see "Using the nocache Parameter" on page 58 for information about the `nocache` parameter.

For more information about `obj.conf`, see the Sun ONE Web Server 6.1 *Administrator's Configuration File Reference*.

find-pathinfo-forward

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

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

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

nostat

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

```
nostat=virtual-path
```

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

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

Using Quality of Service

The quality of service features allow you to limit the amount of bandwidth and number of connections for a server instance, class of virtual servers, or an individual 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 the Sun ONE Web Server 6.1 *Administrator's Guide*.

Using Load Balancing

With load balancing, the amount of server traffic is divided between two or more computers so that more work gets done in the same amount of time and all online users will generally be served faster. Third-party plugins can be used to provide load balancing capabilities for Sun ONE Web Server. Contact load balancing plugin providers for information about solutions that work with Sun ONE Web Server.

Using libloadbal

You can use the load balancing plugin libloadbal to allow your server to execute a program when certain thread load conditions are met, so a load distribution product on the front-end can redistribute the load.

There are two methods that you can use to trigger the load balancer to increase or decrease load, standard or aggressive:

- **Standard**

  Base load decisions on the number of queued requests. This is a passive approach. By letting the queue fill up you are already delaying some requests. In this case, you want the `HighThreshold` to be a low value and `LowThreshold` to be a high value.

- **Aggressive**

  Base load decisions on the number of active threads in the pool. This is designed to more tightly control the requests so that you would reduce the load before requests get queued.

Library configuration

To enable the plugin, you must modify `magnus.conf` manually. This should look something like this:

```
Init fn="load-modules" funcs="init-resonate"
shlib="server_root/bin/https/lib/libloadbal.so"

Init fn="init-resonate" ThreadPool="sleep"
EventExePath="/tools/ns/bin/perl5" LateInit="yes"
CmdLow="/opt/SUNWwbsvr/plugins/loadbal/CmdLow.pl"
CmdHigh="/opt/SUNWwbsvr/plugins/loadbal/CmdHigh.pl"
```

The `init-resonate` function can take the following parameters:

Table 3-1 init-resonate Parameters

Parameter	Description
`ThreadPool`	Name of the thread pool to monitor.
`Aggressive`	If set to `TRUE`, this argument causes the plugin to use the pool thread count rather than the queue thread count.
`PollTime`	How frequently to check the thread status. The default is 2000 milliseconds.
`HighThreshold`	Defines the queue size/# of threads where `HighCmd` is executed to increase load on the server. The default is `4096`.
`LowThreshold`	Defines the queue size/# of threads where the `LowCmd` is executed to decrease load on the server. The default is `1`.
`EventExePath`	Pointer to the script program you want to run (for instance, `/usr/bin/perl` or `/bin/sh`). Defaults to `perl` or `perl.exe`, depending on the platform.
`CmdLow`	Pointer to the script to be run when the `LowThreshold` is met.
`ArgsLow`	Arguments to send to `CmdLow`.
`CmdHigh`	Pointer to the script to be run when the `HighThreshold` is met.
`ArgsHigh`	Arguments to send to `CmdHigh`.

NOTE You must specify `LateInit="yes"` when loading this module. The module creates a monitoring thread, and this monitoring thread must start after `ns-httpd` has started.

If you set `LogVerbose on` in `magnus.conf`, the error log contains information on how the plugin is configured and when it is invoked.

A sample of the information in the error log is shown below:

```
[12/Jun/2003:09:36:35] verbose (20685): Resonate plugin watching
thread pool sleep
[12/Jun/2003:09:36:35] verbose (20685): Resonate plugin
aggressive setting is FALSE
[12/Jun/2003:09:36:35] verbose (20685): Resonate plugin poll time
set to 2000
[12/Jun/2003:09:36:35] verbose (20685): Resonate plugin
HighThreshold set to 5
[12/Jun/2003:09:36:35] verbose (20685): Resonate plugin
LowThreshold set to 1
[12/Jun/2003:09:36:35] verbose (20685): Resonate plugin event
executable path set to /tools/ns/bin/perl5
[12/Jun/2003:09:36:35] verbose (20685): Resonate plugin low
command set to /opt/SUNWwbsvr/plugins/loadbal/CmdLow.pl
[12/Jun/2003:09:36:35] verbose (20685): Resonate plugin high
command set to /opt/SUNWwbsvr/plugins/loadbal/CmdHigh.pl
```

This is what the log entries will look like when `LogVerbose on` is set and the plugin is activated:

```
[12/Jun/2003:09:40:12] verbose (20699): Resonate plugin reducing
load.
[12/Jun/2003:09:40:14] verbose (20699): Resonate plugin reducing
load.
[12/Jun/2003:09:40:16] verbose (20699): Resonate plugin reducing
load.
[12/Jun/2003:09:40:18] verbose (20699): Resonate plugin reducing
load.
[12/Jun/2003:09:40:20] verbose (20699): Resonate plugin reducing
load.
[12/Jun/2003:09:40:30] verbose (20699): Resonate plugin
increasing load.
```

Testing

To test the load balancer, you can create an NSAPI plugin that prints an HTML page and then calls `sleep()` for a period to simulate execution time. This way you can build up a simulated load on the server and ensure that the load balancer commands are working properly.

To configure the sample program:

1. Add a new `mime.type` so this isn't run for every request by modifying `config/mime.types` and adding:

   ```
   type=magnus-internal/sleep exts=sleep
   ```

2. Create a file in your document root directory with the extension of `.sleep`.

   It doesn't matter if anything is in this file; it is used only as a placeholder.

3. Load the module into the server by editing `magnus.conf`.

   ```
   Init fn="load-modules" funcs="dosleep"
   shlib="/opt/SUNWwbsvr/plugins/nsapi/examples/dosleep.so"
   pool="sleep"
   ```

   In the example above, you are changing `shlib` to the location of the library, and setting `pool` to the name of the thread pool you defined earlier.

4. Add this `Service` line where the others are found (note that order is not important):

   ```
   Service method="(GET|HEAD)" fn="dosleep" duration="10"
   type="magnus-internal/sleep"
   ```

   The argument duration tells the server how long to sleep for each request in seconds.

5. Restart your server.

You should now be ready to test the load balancer plugin. The NSAPI plugin will keep the threads busy long enough to simulate your desired load. The load balancing plugin is tested by retrieving the `.sleep` file you created earlier.

Sample

Below is a sample `dosleep.c`:

```
#ifdef XP_WIN32
#define NSAPI_PUBLIC __declspec(dllexport)
#else /* !XP_WIN32 */
#define NSAPI_PUBLIC
#endif /* !XP_WIN32 */
```

```
#include "nsapi.h"

#define BUFFER_SIZE 1024

#ifdef __cplusplus
extern "C"
#endif
NSAPI_PUBLIC int dosleep(pblock *pb, Session *sn, Request *rq)
{
    char buf[BUFFER_SIZE];
    int length, duration;
    char *dur = pblock_findval("duration", pb);

    if (!dur) {
        log_error(LOG_WARN, "dosleep", sn, rq, "Value for duration
is not set.");

        return REQ_ABORTED;
    }

    duration = atoi(dur);

    /* We need to get rid of the internal content type. */
    param_free(pblock_remove("content-type", rq->srvhdrs));
    pblock_nvinsert("content-type", "text/html", rq->srvhdrs);

    protocol_status(sn, rq, PROTOCOL_OK, NULL);

    /* get ready to send page */
    protocol_start_response(sn, rq);

    /* fill the buffer with our message */
    length = util_snprintf(buf, BUFFER_SIZE,
"<title>%s</title><h1>%s</h1>\n", "Sleeping", "Sleeping");
    length += util_snprintf(&buf[length], BUFFER_SIZE - length,
"Sample NSAPI that is sleeping for %d seconds...\n", duration);

    /* write the message to the client */
    if (net_write(sn->csd, buf, length) == IO_ERROR)
    {
        return REQ_EXIT;
    }
    sleep(duration);
    return REQ_PROCEED;
}
```

Chapter 4

Common Performance Problems

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

NOTE For platform-specific issues, see Chapter 5, "Platform-specific Issues and Tips."

Magnus Editor Values

You can set most of the tuning parameter values of the `magnus.conf` file using the Magnus Editor in the Server Manager. However, note that once you have set the values, the Administration Server does not verify that they are valid.

check-acl Server Application Functions

For optimal performance of your server, use ACLs only when required.

The default server is configured with an ACL file containing the default ACL allowing write access to the server only to "all," and an es-internal ACL for restricting write access for "anybody." The latter protects the manuals, icons, and search UI files in the server.

The default `obj.conf` file has `NameTrans` lines mapping the directories that need to be read-only to the es-internal object, which in turn has a `check-acl` SAF for the es-internal ACL.

The default object also contains a `check-acl` SAF for the "default" ACL.

You can improve your server's performance by removing the `aclid` properties from virtual server tags in `server.xml`. This stops any ACL processing.

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

Low-memory Situations

If Sun ONE Web Server must run in low-memory situations, reduce the thread limit to a bare minimum by lowering the value of `RqThrottle`. Also, you may want to reduce the maximum number of processes that the server will spawn by lowering the value of the `MaxProcs` value.

Your web applications running under stress may sometimes result in the server running out of Java VM runtime heap space, as can be seen by `java.lang.OutOfMemoryError` messages in the server log file. There could be several reasons for this (such as excessive allocation of objects), but such behavior could affect performance. To address this problem, you would need to 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/PerformanceFAQ.html`

At times your application could be running out of maximum sessions (as evidenced by the "too many active sessions message" in the server log file), which would result in the container throwing exceptions, which in turn impacts application performance. A due consideration of session manager properties, session creation activity (note that JSPs have session enabled by default), and session idle time would be needed to address this situation.

Under-throttled Server

The server does not allow the number of active threads to exceed the thread limit value. If the number of simultaneous requests reaches that limit, the server stops servicing new connections until the old connections are freed up. This can lead to increased response time.

In Sun ONE Web Server, the server's default `RqThrottle` value is 128. If you want your server to process more requests concurrently, you need to increase the `RqThrottle` value.

The symptom of an under-throttled server is a server with a long response time. Making a request from a browser establishes a connection fairly quickly to the server, but on under-throttled servers it may take a long time before the response comes back to the client.

The best way to tell if your server is being throttled is to see if the number of active sessions is close to, or equal to, the maximum number allowed via `RqThrottle`. To do this, see "Maximum Simultaneous Requests" on page 49.

Cache Not Utilized

If the cache is not utilized, your server is not performing optimally. Since most sites have lots of GIF or JPEG files that should always be cacheable, you need to use your cache effectively.

Some sites, however, do almost everything through CGIs, SHTML, or other dynamic sources. Dynamic content is generally not cacheable, and inherently yields a low cache hit rate. Don't be too alarmed if your site has a low cache hit rate. The most important thing is that your response time is low. You can have a very low cache hit rate and still have very good response time. As long as your response time is good, you may not care that the cache hit rate is low.

Check your hit ratio using statistics from `perfdump` or the Monitor Current Activity page of the Server Manager. The hit ratio is the percentage of times the cache was used with all hits to your server. A good cache hit rate is anything above 50%. Some sites may even achieve 98% or higher. For more information, see "Cache Information" on page 37.

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

Keep-Alive Connections Flushed

A web site that might be able to service 75 requests per second without keep-alive connections may be able to do 200-300 requests per second when keep-alive is enabled. Therefore, as a client requests various items from a single page, it is important that keep-alive connections are being used effectively. If the `KeepAliveCount` exceeds the `MaxKeepAliveConnections`, subsequent keep-alive connections will be closed, or "flushed," instead of being honored and kept alive.

Check the `KeepAliveFlushes` and `KeepAliveHits` values using statistics from `perfdump` or the Monitor Current Activity page of the Server Manager. For more information, see "Keep-Alive/Persistent Connection Information" on page 33.

On a site where keep-alive connections are running well, the ratio of `KeepAliveFlushes` to `KeepAliveHits` is very low. If the ratio is high (greater than 1:1), your site is probably not utilizing keep-alive connections as well as it could.

To reduce keep-alive flushes, increase the `MaxKeepAliveConnections` value in the `magnus.conf` file or the Magnus Editor of the Server Manager. The default value is 256. By raising the value, you keep more waiting keep-alive connections open.

CAUTION On UNIX/Linux systems, if the `MaxKeepAliveConnections` 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.

Log File Modes

Keeping the log files on verbose mode can have a significant impact on performance. You can set `LogVerbose` to "on" in `magnus.conf` or the Magnus Editor of the Server Manager.

Platform-specific Issues and Tips

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

Solaris-specific Issues

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

Files Open in a Single Process

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 to 8192. To do so on Solaris, make the change in the file `/etc/system` by setting `rlim_fd_max`, and then rebooting.

File Descriptor Limits

Append the following line to the file `/etc/system` to increase the number of file descriptors for Sun ONE Web Server:

```
set rlim_fd_max=65536
```

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, any line added to `/etc/system` should be removed and added again only after verifying that it is still valid.

Failure to Connect to HTTP Server

If users are experiencing connection timeouts from a browser to Sun ONE 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 file `ListenQ` parameter in the `magnus.conf` file:

```
ListenQ  8192
```

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 they are not zero, adjust the value to 2048 initially, and continue to monitor the `netstat` output.

The Sun ONE Web Server `ListenQ` setting and the related Solaris `tcp_conn_req_max_q` and `tcp_conn_req_max_q0` settings should match the throughput of the Sun ONE Web Server HTTP 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 Sun ONE Web Server HTTP server application.

You don't want to accept more connections than the Sun ONE Web Server HTTP server will be able to process. It is better to 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 will typically reduce 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 it 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, it is suggested that you do not change the above parameter.

Tuning TCP Buffering

If you are seeing unpredictable intermittent slowdowns in network response from a consistently loaded server, you might 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 available on Solaris 8 Update 5 and higher.

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)`

Additional information about the configuration and tuning of SNCA for a particular operating system version and patch level can also be obtained from the many resources on `http://docs.sun.com`.

To enable SNCA to work with Sun ONE Web Server (assuming SNCA configuration, as discussed above):

1. Edit the Sun ONE Web Server `server.xml` file so the listen socket on port 80 includes `family="nca"` as shown below (the server must be listening on port 80 for this to work):

   ```
   <LS id="ls1" ip="0.0.0.0" port="80" family="nca" security="off"
   acceptorthreads="1">
   ```

2. Edit the Sun ONE Web Server `nsfc.conf` file and set the following:

   ```
   CacheFileContent=false
   TransmitFile=true
   ```

3. Restart the Web Server for changes to take effect.

RqThrottle and ConnQueueSize

When configuring Sun ONE Web Server to be used with SNCA, setting the `RqThrottle` and `ConnQueueSize` parameters in `magnus.conf` 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..

NOTE `RqThrottle` and `ConnQueueSize` must *both* be set to 0.

For more information about `RqThrottle` and `ConnQueueSize`, see the chapter pertaining to `magnus.conf` in the Sun ONE Web Server 6.1 *Administrator's Configuration File Reference*. Also consult the `RqThrottle` and `ConnQueueSize` entries in the index in this book.

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
- Reduce File System Housekeeping
- Long Service Times on Busy Disks or Volumes

Please 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, you may benefit from increasing 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 will map into its address space for the file system cache. The default value is 12; that is, the kernel will reserve 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 should experiment with this value, starting with values around 25. On systems with large amounts of physical memory, you should 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 could turn off the same by adding the `noatime` parameter to the data volume's mount point in `/etc/vfstab`. For example:

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

Long Service Times on Busy Disks or Volumes

Sun ONE 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 may help Sun ONE Web Server performance substantially.

Your first step should be 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 will usually give a far greater payoff than trying to tune the overloaded disks.

Solaris-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
- Long-term System Monitoring
- "Intelligent" Monitoring

The tools described in this section monitor performance from the standpoint of how the system responds to the load Sun ONE Web Server generates. For information about using Sun ONE Web Server's own capabilities to track the demands users place on the Web Server itself, see “Monitoring Server Performance” on page 17.

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, and for any disk busy more than about 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, you may need to 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 may help you figure out what has changed if the system starts behaving poorly. We recommend you 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.
- Run the command `crontab -e sys` and remove the `#` comment characters from the lines with the `sa1` and `sa2` commands. You may also wish to adjust how often the commands run and at what times of day depending on your site's activity profile (see `man crontab` for an explanation of the format of this file).

This 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 may need adjustment. You can download the toolkit and its documentation from the following location:

`http://www.setoolkit.com/`

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 might tune your system to achieve the desired result.

Table 5-1 Tuning Solaris for performance benchmarking

Parameter	Scope	Default Value	Tuned Value	Comments
rlim_fd_max	/etc/system	1024	8192	Process open file descriptors limit; should account for the expected load (for the associated sockets, files, pipes if any).
rlim_fd_cur	/etc/system	64	8192	
sq_max_size	/etc/system	2	0	Controls streams driver queue size; setting to 0 makes it infinity so the performance runs won't be hit by lack of buffer space. Set on clients too.
tcp_close_wait_interval	ndd /dev/tcp	240000	60000	Set on clients too.
tcp_time_wait_interval	ndd /dev/tcp	240000	60000	For Solaris 7 only. 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%, you should increase this value.
tcp_rexmit_interval_max	ndd /dev/tcp	240000	10000	
tcp_rexmit_interval_min	ndd /dev/tcp	200	3000	

Table 5-1 Tuning Solaris for performance benchmarking

Parameter	Scope	Default Value	Tuned Value	Comments
`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_recv_hiwat`	`ndd /dev/tcp`	8129	32768	To increase the receive buffer.

Chapter 6

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:

Processors

On Solaris and Windows, Sun ONE 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 any busy CPU activity.

Memory

As a baseline, Sun ONE Web Server requires 64 MB RAM. Multiple CPUs require at least 64 MB per CPU. For example, if you have four CPUs, you should 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 per 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, Sun ONE Web Server logs, and document tree each on separate hard drives. Thus, if your log files fill up the log drive, your OS will not suffer. Also, you'll be able to tell whether, for example, the OS paging file is causing drive activity.

Your OS vendor may have specific recommendations for how much swap or paging space you should allocate. Based on our testing, Sun ONE 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 may include multiple documents. If you're 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. That's the WAN bandwidth your server needs.

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

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

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

Chapter 7

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 might configure your system to best take advantage of Sun ONE Web Server's strengths.

This chapter includes the following topics:

Study Goals

This study shows how well Sun ONE Web Server 6.1 scales against 1, 2, and 4 CPUs. The goal of the tests in the study was to saturate the server CPU. The tests also help to determine what kind of configuration (CPU and memory) is required for different types of content. The studies were conducted against the following content:

- 100% static
- 100% C CGI
- 100% Perl CGI
- 100% NSAPI
- 100% Java servlets
- 100% PHP/FastCGI

General Conclusions

- The tuned server performed significantly better than the out-of-the-box server for static loads.
- The tuned server performed slightly better than the out-of-the-box server for dynamic loads.
- The tuned server showed no significant performance improvement for SSL-encrypted static and dynamic workloads.

Sun ONE Web Server Configuration

- Mostly out-of-the box settings
- File cache configured via `nsfc.conf` for cache static tests
- Tested with two virtual servers (secure and non-secure) on two listen sockets of the same instance
- SSL and non-SSL run without configuring two instances
- Java tests run with both the default and `/usr/lib/lwp` thread libraries
- HTTP/1.0 and HTTP/1.1 for static tests

This section lists:

- Tuned Server Settings
- nsfc.conf Settings
- System Configuration

Tuned Server Settings

The following table shows the server settings for the non-SSL performance runs. Also note the following:

- `nsfc.conf` in the tuned server was configured to cache all files in the heap.
- The size of files in the specweb fileset ranged between 102 bytes to 912 KB.
- JVM settings were default.

Table 7-1 Tuned Server Settings

Setting	Value
DNS	Off
`AccessLog`	Off
`StackSize`	262144
`MaxKeepAliveConnections`	2000
`ConnQueueSize`	10000
`ListenQ`	8192
`SSLCacheEntries`	100000000
`SSL3SessionTimeout`	86400
`SSLSessonTimeout`	100
`CGIStubIdleTimeout`	10000

nsfc.conf Settings

The following table lists the `nsfc.conf` settings.

Table 7-2 nsfc.conf Settings

Setting	Out of the Box	Tuned
`MaxAge`	30	86400
`MaxFiles`	1024	240000
`SmallFileSizeLimit`	2048	1000000
`SmallFileSpace`	1048576	2147483648
`MediumFileSizeLimit`	537600	1000001
`MediumFileSpace`	10485760	1

For more information about `nsfc.conf`, see the Sun ONE Web Server 6.1 *Administrator's Configuration File Reference*.

System Configuration

- Server machine: Sun Fire™ V880, 900 Mhz (only 4 CPUs were used for the tests)
- RAM: 16384 MB
- Network connection: 1 GB/sec private interface

Performance Results

For most cases, scalability plots are shown. Performance is shown as a function of the number of CPUs enabled. 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

While operations per second (ops/sec) data is shown for most cases, the response time and throughput are shown only where available.

Results are provided for the following tests, which are discussed in the remainder of this chapter:

Static Content Test

This test was performed with static download of a randomly selected file from a pool of 400 directories, each containing 100 files ranging in size from 5 KB to 250 KB. Tests were done with the file cache configured to include all files in the directories. The goal of static content tests was to identify the maximum number of conforming connections the server could handle. A conforming connection is one that operates faster than 320 Kbps (kilobits per second).

Simultaneous connections: 1500

Figure 7-1 Static Content Test

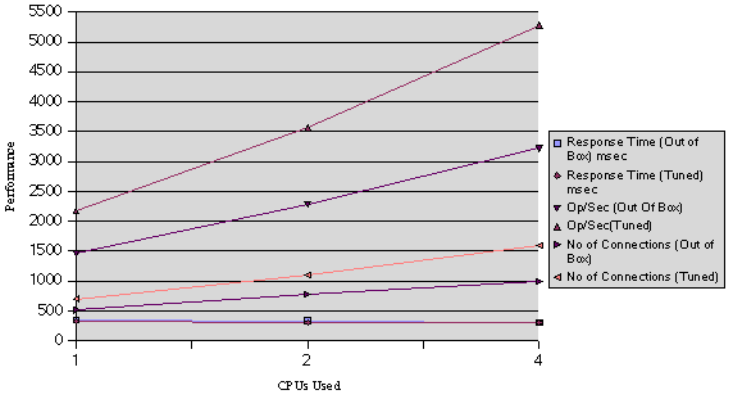

Table 7-3 Static Content Test

CPUs	Response Time (Out of Box) msec	Response Time (Tuned) msec	Op/Sec (Out of Box)	Op/Sec (Tuned)	Number of Connections (Out of Box)	Number of Connections (Tuned)
1	346.69	320.5	1456.9	2169.3	510	700
2	337.01	305.3	2280.1	3565.1	775	1100
4	307.19	299.6	3220.8	5279.1	1000	1600

Dynamic Content Test: WASP Servlet

This test was conducted using the WASP servlet. It prints out the servlet's initialization arguments, environments, request headers, connection/client info, URL information, and remote user information. The goal was to saturate the CPUs on the server.

Number of clients: 3600

Figure 7-2 Dynamic Content Test: WASP Servlet

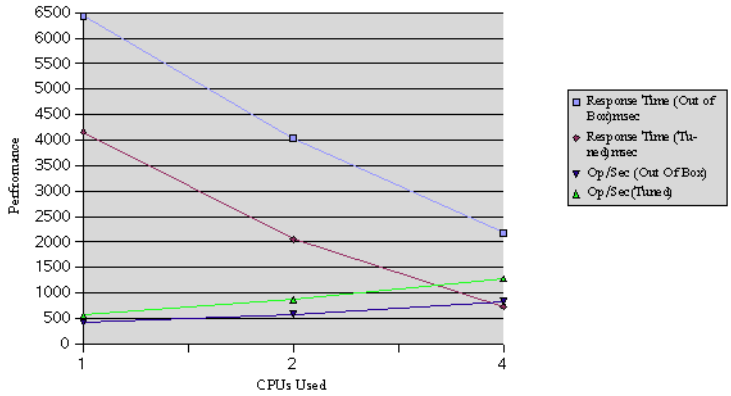

Table 7-4 Dynamic Content Test: WASP Servlet

CPUs	Response Time (Out of Box) msec	Response Time (Tuned) msec	Op/Sec (Out of Box)	Op/Sec (Tuned)
1	6436.46	4159.93	414.6	571.87
2	4031.66	2052.63	518.8	870.25
4	2177.81	732.42	832.1	1280.43

Dynamic Content Test: C CGI

This test was performed by accessing a `C` executable called `printenv`. This executable outputs approximately 0.5 KB of data per request. The goal was to saturate the CPUs on the server.

Number of clients: 2400

Figure 7-3 Dynamic Content Test: C CGI

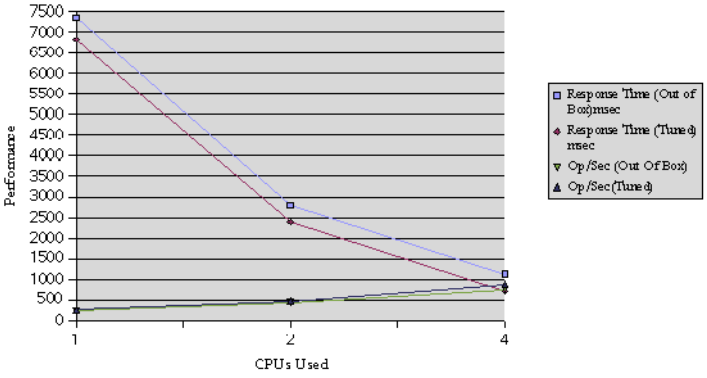

Table 7-5 Dynamic Content Test: C CGI

CPUs	Response Time (Out of Box) msec	Response Time (Tuned) msec	Op/Sec (Out of Box)	Op/Sec (Tuned)
1	7350.41	6819.63	244.8	265.17
2	2801.64	2391.25	436.8	473.46
4	1127.31	719.36	750.59	873.6

Dynamic Content Test: Perl CGI

This test ran against a Perl script called `printenv.pl` that prints the CGI environment. This script outputs approximately 0.5 KB of data per request. The goal was to saturate the CPUs on the server.

Number of clients: 450

Figure 7-4 Dynamic Content Test: Perl CGI

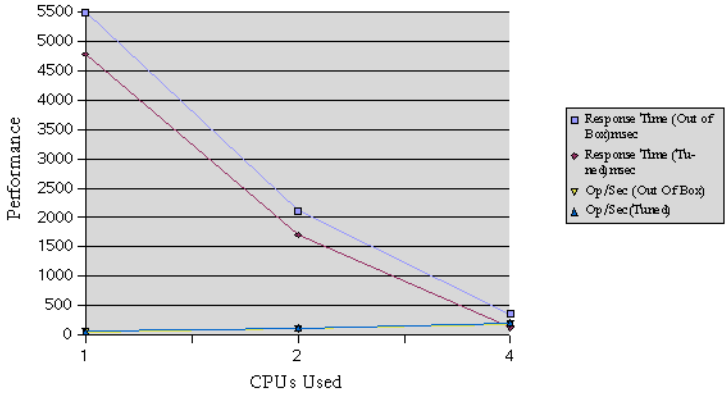

Table 7-6 Dynamic Content Test: Perl CGI

CPUs	Response Time (Out of Box) msec	Response Time (Tuned) msec	Op/Sec (Out of Box)	Op/Sec (Tuned)
1	5484.17	4777.72	57.6	62.05
2	2111.22	1704.28	107.8	119.32
4	363.81	132.85	189.6	209.76

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 CPUs on the server.

Number of clients: 6300

Figure 7-5 Dynamic Content Test: NSAPI

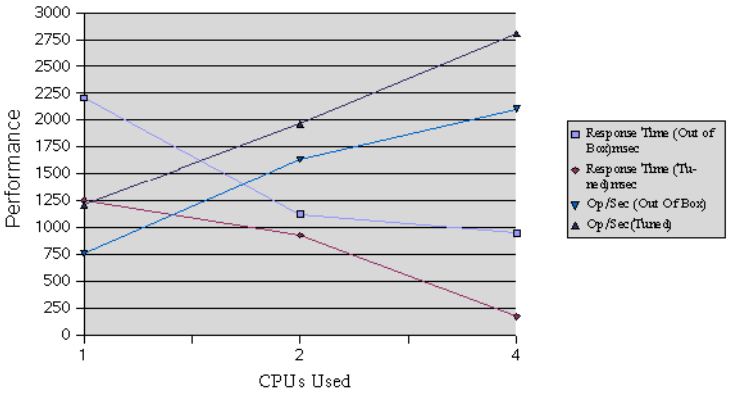

Table 7-7 Dynamic Content Test: NSAPI

CPUs	Response Time (Out of Box) msec	Response Time (Tuned) msec	Op/Sec (Out of Box)	Op/Sec (Tuned)
1	2208.06	1259.16	758.9	1212.07
2	1123.85	931.13	1636.3	1965.68
4	952.67	177.9	2106.1	2804.05

SSL Performance Test: Static Content

A 1 KB static SSL file was used for this test. The goal was to saturate the CPUs on the server.

Simultaneous connections: 550

Figure 7-6 SSL Test: Static Content

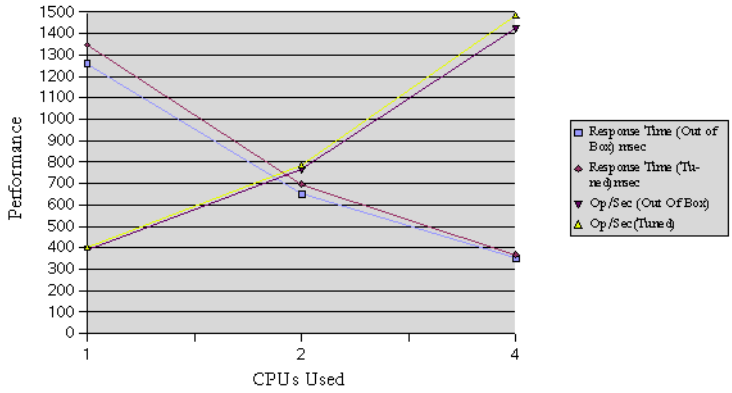

Table 7-8 SSL Test: Static Content

CPUs	Response Time (Out of Box) msec	Response Time (Tuned) msec	Op/Sec (Out of Box)	Op/Sec (Tuned)
1	1259.11	1357.81	392.5	404.7
2	650.61	697.31	764.3	784.3
4	351.31	368.01	1422.6	1484.5

SSL Performance Test: Perl CGI

This test was performed by accessing the `printenv C` executable in SSL mode. The goal was to saturate the CPUs on the server. The test was performed in SSL mode with the SSL session cache both enabled and disabled.

Figure 7-7 SSL Test: Perl CGI

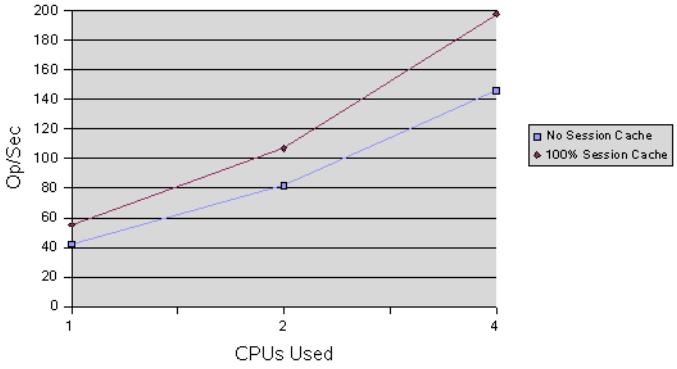

Table 7-9 SSL/Perl CGI: No Session Cache Reuse

# of CPUs	Op/Sec (Out of Box)	Op/Sec (Tuned)
1	41.9	42.19
2	81.0	81.86
4	145.1	146.05

Table 7-10 SSL/Perl CGI: 100% Session Cache Reuse

# of CPUs	Op/Sec (Out of Box)	Op/Sec (Tuned)
1	55.29	55.42
2	105.01	107.05
4	194.35	197.91

Table 7-11 SSL/Perl CGI: Session Cache Comparison

# of CPUs	No Session Cache (Tuned)	100% Session Cache (Tuned)
1	42.19	55.42
2	81.86	107.05
4	146.05	197.91

SSL Performance Test: C CGI

This test was performed by accessing the `printenv C` executable in SSL mode. The goal was to saturate the CPUs on the server. The test was performed in SSL mode with the SSL session cache both enabled and disabled.

Figure 7-8 SSL Test: C CGI

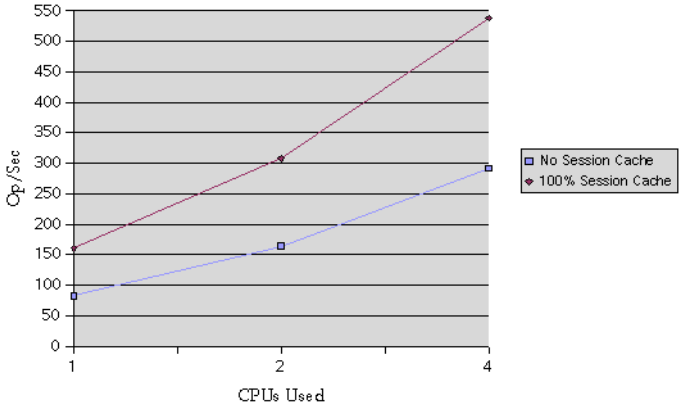

Table 7-12 SSL/C CGI: No Session Cache Reuse

CPUs	Op/Sec (Out of Box)	Op/Sec (Tuned)
1	84.8	82.73
2	165.0	164.38
4	290.6	291.63

Table 7-13 SSL/C CGI: 100% Session Cache Reuse

CPUs	Op/Sec (Out of Box)	Op/Sec (Tuned)
1	160.65	165.69
2	308.11	310.51
4	538.54	550.19

Table 7-14 SSL/C CGI: Session Cache Comparison

CPUs	No Session Cache (Tuned)	100% Session Cache (Tuned)
1	82.73	160.65
2	164.38	308.11
4	291.63	538.54

SSL Performance Test: NSAPI

This test was performed by accessing the `printenv C` executable in SSL mode. The goal was to saturate the CPUs on the server. The test was performed in SSL mode with the SSL session cache both enabled and disabled.

Figure 7-9 SSL Test: NSAPI

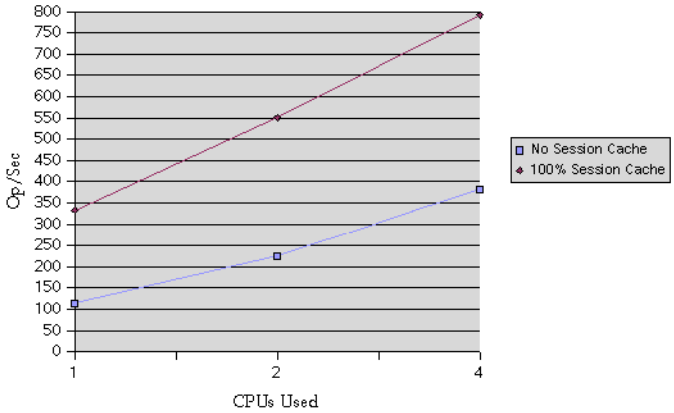

Table 7-15 SSL/NSAPI: No Session Cache Reuse

CPUs	Op/Sec (Out of Box)	Op/Sec (Tuned)
1	114.08	114.44
2	223.58	225.04
4	380.88	382.78

Table 7-16 SSL/NSAPI: 100% Session Cache Reuse

CPUs	Op/Sec (Out of Box)	Op/Sec (Tuned)
1	321.24	333.21
2	554.87	551.45
4	762.04	791.62

Table 7-17 SSL/NSAPI: Session Cache Comparison

CPUs	No Session Cache (Tuned)	100% Session Cache (Tuned)
1	114.44	333.21
2	225.04	551.45
4	382.78	791.62

JDBC Connection Pooling with OCI Driver

This test tested the scalability and performance of the JDBC connection pooling module. In this test a simple servlet requests a row from a large database and prints its content. An Oracle database and the Oracle OCI driver were used for the test. JDBC connection pool resource configuration is shown below (`server.xml`).

```
<RESOURCES>

       <JDBCRESOURCE jndiname="jdbc/tpcwDB" poolname="TpcwPool"
enabled="true"/>
              <JDBCCONNECTIONPOOL name="TpcwPool"
datasourceclassname="oracle.jdbc.pool.OracleDataSource"
steadypoolsize="1000" maxpoolsize="1000" poolresizequantity="2"
idletimeout="0" maxwaittime="0"
connectionvalidationrequired="false"
connectionvalidationmethod="auto-commit"
validationtablename="string" failallconnections="false" >
    <PROPERTY name="URL"
value="jdbc:oracle:oci8:@(description=(address=(host=mach-3)
(protocol=tcp)(port=1521))(connect_data=(sid=10K)))"/>
 <PROPERTY name="user" value="tpcw"/>
 <PROPERTY name="password" value="tpcw"/>
              </JDBCCONNECTIONPOOL>
 </RESOURCES>
```

Number of clients: 3600

Figure 7-10 JDBC Connection Pooling Test

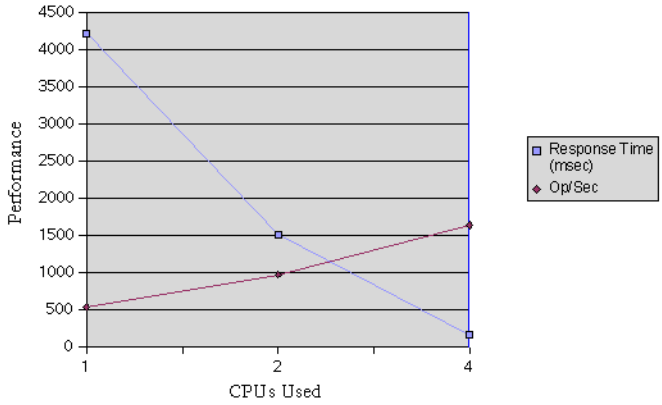

Table 7-18 JDBC Connection Pooling Test

CPUs	Response Time (msec)	Op/Sec
1	4223.66	529.14
2	1508.53	966.74
4	153.19	1634.94

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 Sun ONE Web Server combined with the versatility of the PHP engine provides a highly performant and versatile web deployment platform for dynamic content.

The PHP (version 4.3.2) tests were performed in two modes:

- Out-of-process "fastcgi-php" application invoked using the FastCGI plugin available for Sun ONE Web Server 6.1 (the download will be available from `http://www.zend.com/sun/)`.
- In-process PHP NSAPI plugin (available with PHP).

The test executes the `phpinfo()` query. Mostly out-of-the-box settings were used, plus PHP-related settings in the `obj.conf` and `magnus.conf` files, as shown after the test graphs and data.

FastCGI

Figure 7-11 PHP Scalability Test: FastCGI

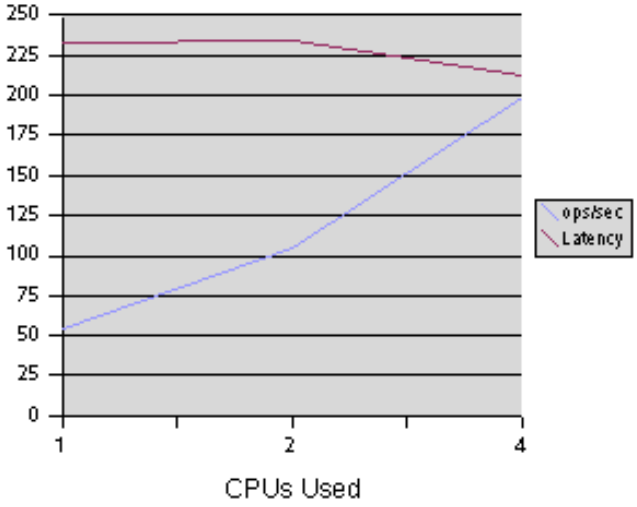

Table 7-19 PHP Scalability Test: FastCGI

CPUs	Op/Sec	Latency (msec)
1	54	214
2	105	225
4	199	230

NSAPI

Figure 7-12 PHP Scalability Test: NSAPI

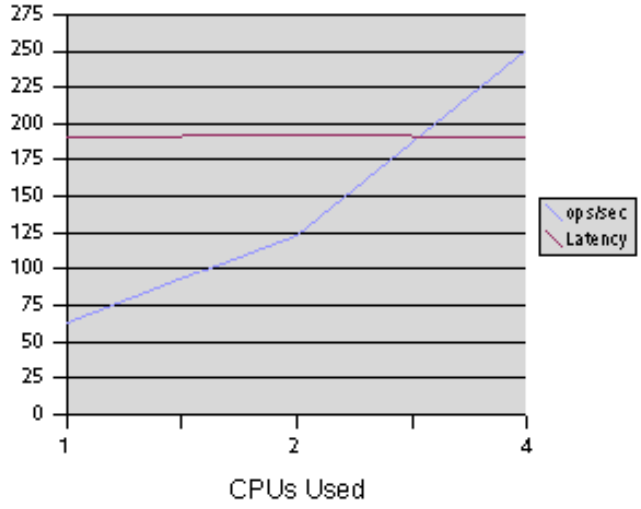

Table 7-20 PHP Scalability Test: NSAPI

CPUs	Op/Sec	Latency
1	63	190
2	125	193
4	251	190

magnus.conf Settings

```
Init fn="load-modules"
shlib="/export0/ES61/install/bin/https/lib/libphp4.so"\
funcs="php4_init,php4_close,php4_execute,php4_auth_trans"
Init fn="php4_init"\
      errorString="PHP Totally Blowed Up!"

Init fn="load-modules"
shlib="/export0/ES61/install/bin/https/lib/libnsapi_fcgi.
so" funcs="FCGIRequestHandler,FCGIInit"
shlib_flags="(global|now)"

Init fn="FCGIInit" errorString "Unable to start the FCGI NSAPI
module"
```

obj.conf Settings

```
NameTrans fn="pfx2dir"
from="/php-nsapi"dir="/export0/ES61/install/docs/php-nsapi"
name="php-nsapi"
NameTrans fn="pfx2dir"
from="/php-fcgi"dir="/export0/ES61/install/docs/php-fcgi"
name="fastcgi"

Service type="magnus-internal/fastcgi-php"
fn="FCGIRequestHandler"
BindPath="localhost:8082" AppPath="/export0/php-fastcgi/bin/php"
StartServers="5" PHP_FCGI_CHILDREN="10"
PHP_FCGI_MAX_REQUEST="2000"

<Object name="fastcgi">
ObjectType fn="force-type" type="magnus-internal/fastcgi-php"
Service type="magnus-internal/fastcgi-php"\
         fn=FCGIRequestHandler\
         BindPath="localhost:8082"\
         AppPath="/export0/php-fastcgi/bin/php"
         StartServers="5"\
         PHP_FCGI_CHILDREN="10"\
         PHP_FCGI_MAX_REQUEST="2000"
</Object>

<Object name="php-nsapi">
# Set the MIME type
ObjectType fn="force-type" type="magnus-internal/x-httpd-php"
# Run the function
Service fn=php4_execute
</Object>
```

Index

D

F

H

I

J

K

L

M

N

O

P

Q

R

S

T

U

V

W