Sun Java System Web Proxy Server 4.0.12 Performance Tuning, Sizing, and Scaling Guide

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

Part No: 821–0568
September 2009

091015@23031

Contents

Tables

Figures

Preface

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

Who Should Use This Book

This guide is intended for advanced administrators only. Be sure to read this guide and other relevant server documentation before making any changes.

Before You Read This Book

Proxy Server can be installed as a stand-alone product or as a component of Sun Java Enterprise System (Java ES), a software infrastructure that supports enterprise applications distributed across a network or Internet environment. If you are installing Proxy Server as a component of Java ES, you should be familiar with the system documentation at `http://docs.sun.com/coll/1286.2`.

How This Book Is Organized

This guide is divided into chapters. Each chapter addresses specific areas and tasks. The following table lists the chapters of the guide and their contents.

TABLE P–1 Guide Organization

Chapter	Description
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 Java System Web Proxy Server"	This chapter describes specific adjustments you can make that might improve Sun Java System Web Proxy Server performance. It provides an overview of Proxy Server's connection-handling process so that you can better understand the tuning settings.
Chapter 3, "Common Performance Problems"	This chapter discusses common web site performance problems.
Chapter 4, "Platform-Specific Issues and Tips"	This chapter provides platform-specific tuning tips.
Chapter 5, "Sizing and Scaling Your Server"	This chapter examines the subsystems of your server, and provides recommendations for optimal performance.
Chapter 6, "Scalability Studies"	This chapter describes the results of scalability studies.

Proxy Server Documentation Set

The Proxy Server documentation set describes how to install and administer the Proxy Server. The documentation set lists the documents that are related to Proxy Server. The URL for Sun Java System Web Proxy Server 4.0.12 documentation is `http://docs.sun.com/coll/1311.12`. For an introduction to Proxy Server, refer to the books in the order in which they are listed in the following table.

TABLE P–2 Sun Java System Web Proxy Server Documentation

Document Title	**Contents**
Sun Java System Web Proxy Server 4.0.12 Release Notes	The Proxy Server release: ■ Late-breaking information about the software and the documentation ■ New features ■ Supported platforms and environments ■ System requirements ■ Known issues and workarounds

TABLE P–2 Sun Java System Web Proxy Server Documentation *(Continued)*

Sun Java System Web Proxy Server 4.0.12 Installation and Migration Guide	Performing installation and migration tasks: ■ Installing the Sun Java System Web Proxy Server ■ Migrating from version 3.6 to version 4
Sun Java System Web Proxy Server 4.0.12 Administration Guide	Performing administration and management tasks: ■ Using the administration and command-line interfaces ■ Configuring server preferences ■ Managing users and groups ■ Monitoring and logging server activity ■ Using certificates and public key cryptography to secure the server ■ Controlling server access ■ Proxying and routing URLs ■ Caching ■ Filtering content ■ Using a reverse proxy ■ Using SOCKS
Sun Java System Web Proxy Server 4.0.12 Configuration File Reference	Editing configuration files
Sun Java System Web Proxy Server 4.0.12 NSAPI Developer's Guide	Creating custom Netscape Server Application Programmer's Interface (NSAPI) plugins
Sun Java System Web Proxy Server 4.0.12 Performance Tuning, Sizing, and Scaling Guide	Tuning Sun Java System Web Proxy Server to optimize performance

Related Books

The URL for all documentation about Sun Java Enterprise System (Java ES) and its components is `http://docs.sun.com/prod/entsys.5`.

Default Paths and File Names

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

TABLE P–3 Default Paths and File Names

Placeholder	Description
install-dir	Represents the base installation directory for Proxy Server

TABLE P–3 Default Paths and File Names *(Continued)*

Placeholder	Description
instance-dir	Directory that contains the instance-specific subdirectories

Typographic Conventions

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

TABLE P–4 Typographic Conventions

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

Symbol Conventions

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

TABLE P–5 Symbol Conventions

Symbol	Description	Example	Meaning
`[ ]`	Contains optional arguments and command options.	`ls [-l]`	The `-l` option is not required.
`{ | }`	Contains a set of choices for a required command option.	`-d {y|n}`	The `-d` option requires that you use either the `y` argument or the `n` argument.
`${ }`	Indicates a variable reference.	`${com.sun.javaRoot}`	References the value of the `com.sun.javaRoot` variable.

TABLE P–5 Symbol Conventions *(Continued)*

Symbol	Description	Example	Meaning
-	Joins simultaneous multiple keystrokes.	Control-A	Press the Control key while you press the A key.
+	Joins consecutive multiple keystrokes.	Ctrl+A+N	Press the Control key, release it, and then press the subsequent keys.
→	Indicates menu item selection in a graphical user interface.	File → New → Templates	From the File menu, choose New. From the New submenu, choose Templates.

Documentation, Support, and Training

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

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

Searching Sun Product Documentation

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

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

For example, to search for "Proxy Server," type the following:

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

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

Third-Party Web Site References

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

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

Sun Welcomes Your Comments

Sun is interested in improving its documentation and welcomes your comments and suggestions. To share your comments, go to `http://docs.sun.com` and click Feedback.

CHAPTER 1

Performance and Monitoring Overview

Sun Java™ System Web Proxy Server software (known as Proxy Server) is designed to meet the needs of high-traffic sites in the world. It can serve both static and dynamically generated content. Proxy Server can run in the Secure Sockets Layer (SSL) mode, enabling 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, so the impact of the suggestions provided in this guide will depend on your specific environment.

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

This chapter includes the following topics:

- "Performance Issues" on page 17
- "SSL Performance" on page 18
- "Monitoring Server Performance" on page 19

Performance Issues

You must first determine your requirements. Users want fast response times, typically less than 100 milliseconds, high availability with no connection refused messages, and significant control. Webmasters and proxy server administrators, on the other hand, need high connection rates, high data throughput, and uptime approaching 100%. You need to define what performance means for your particular situation based on your requirement.

The following factors have an impact on performance:

- The Number of peak concurrent users
- Security requirements

Encrypting your proxy server's data streams with SSL makes an enormous difference to your site's credibility for electronic commerce and other security conscious applications, but can seriously impact your CPU load. For more information, see "SSL Performance" on page 18.

- Disk Cache hits or misses

  A high percentage of cache hits indicate efficient utilization of cached objects, which in turn leads to improved performances.
- Custom configurations to increase cache efficiency

  You can configure Proxy Server to increase cache efficiency, and performance, at the cost of spec-compliance. For example, you can have configurations that ignores page reload requests , or those that ignore cache directives in response headers that do not allow caching the response.
- Disk cache location

  Using RAM-based file systems to hold the disk cache can significantly improve performance.
- Hardware bottlenecks

  Care should be taken to ensure that hardware factors such as network speed and disk throughput match the processing power of the CPU as well as the request handling capacity of the proxy server.
- Behavior of origin servers

  The origin servers response time has a crucial effect on the proxy server's performance numbers.

SSL Performance

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

For SSL, Proxy Server uses the Network Security Services (NSS) library. However, you can use other options for SSL:

- If you are using the Solaris™ 10 operating system, kernel SSL (KSSL) is available. It does not contain all the algorithms displayed, as does NSS, but it often provides better performance.
- A cryptographic card hardware accelerator for SSL can also improve performance.

Monitoring Server Performance

You must measure the system behavior before and after a change to check performance. You can monitor the performance of Proxy Server in different ways.

TABLE 1–1 Methods of Monitoring Performance

Monitoring Method	How to Enable	How to Access	Advantages and Requirements
Statistics through the Admin console	Enabled by default	In the Admin console, for a configuration, click the Monitor tab	Accessible when session threads are hanging. Administration Server must be running.
XML-formatted statistics (`stats-xml`) by using a browser	Enable through Admin console or by editing a configuration file	Through a URI	Administration Server need not be running.
`perfdump` by using a browser	Enable through Admin console or by editing a configuration file	Through a URI	Administration Server need not be running.
Java ES monitoring	Enabled by default	Through the Java ES Monitoring Console	Only for Java ES installations. Administration Server must be running.

Monitoring the server does have some impact on computing resources. In general, using `perfdump` through the URI is the least expensive, followed by using `stats-xml` through a URI. Because using the Administration Server requires computing resources, using the Admin console is an expensive monitoring method.

For more information about the monitoring methods, see the following sections:

- "About Statistics" on page 20
- "Monitoring Current Activity Using `stats-xml`" on page 21
- "Monitoring Current Activity Using `perfdump`" on page 22
- "Monitoring Current Activity Using the Java ES Monitoring Console" on page 24

About Statistics

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

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

For configuration, statistics are available in the following areas:

- Requests
- Errors
- Response Time

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

- Requests
- Errors
- Response Time
- General
- Java Virtual Machine (JVM™)
- Connection Queue
- Keep Alive
- Host DNS Cache
- Client DNS Cache
- In-Memory File Cache
- Thread Pools
- Session Threads, including profiling data (exists if profiling is enabled)

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

Enabling Statistics

To enable statistics, use Admin Console.

Note – Collecting statistics causes a slight hit to performance.

▼ To Enable Statistics (`stats-xml`) from the Admin Console

1. **Select the Proxy Server instance.**
2. **Click the Server Status tab.**
3. **Click the Monitor Current Activity sub tab.**
4. **Choose Yes for Activate Statistics/Profiling?**
5. **Save and apply changes.**

Monitoring Current Activity Using `stats-xml`

You can display statistics in XML format by using `stats-xml`. You can view the `stats-xml` output through a URI, that you need to enable, or you can view the `stats-xml` output through the CLI, that is enabled by default.

▼ To Monitor Current Activity from the Admin Console

1. **Select the Proxy Server instance.**
2. **Click the Server Status tab.**
3. **Click the Monitor Current Activity sub tab.**

   Ensure that Statistics is enabled (see above).
4. **Select the required Statistics from the dropdown list under Monitor Proxy Server Statistics and click Submit.**

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

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

- **Modify the `stats-xml` URI to limit the information by setting elements to `0` or `1`.**

  An element set to `0` is not displayed on the `stats-xml` output. For example:

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

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

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

Use the following syntax elements to limit `stats-xml`statistics:

- `cache-bucket`
- `connection-queue`
- `connection-queue-bucket` (process-level)
- `cpu-info`
- `host-dns-bucket`
- `client-dns-bucket`
- `keepalive-bucket`
- `process`
- `profile`
- `profile-bucket` (process-level)
- `request-bucket`
- `thread`
- `thread-pool`
- `thread-pool-bucket` (process-level)

Monitoring Current Activity Using `perfdump`

To Enable and Use the `perfdump` SAF

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 line to the default object:**

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

3. **Restart your server software.**

4. **Go to `http://computer_name:proxyport/.perf` and access `perfdump`.**

   You can specify the request time for the `perfdump` statistics. The browser automatically refreshes the statistics based on the time you specify. The following example sets the refresh time to every 5 seconds:

```
http://computer_name:proxyport/.perf?refresh=5
```

Using Performance Buckets

Performance buckets enable 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 the data to the bucket. The cost of collecting this information is minimal, and the impact on the server performance is usually negligible. You can access this information by using `perfdump`. The following information is stored in a bucket:

- **Name of the bucket.** This name associates the bucket with a function.
- **Description.** A description of the functions with which the bucket is associated.
- **Number of requests for this function.** The total number of requests that caused this function to be called.
- **Number of times the function was invoked.** This number might not coincide with the number of requests for the function, because some functions might be executed more than once for a single request.
- **Function latency or the dispatch time.** The time taken by the server to invoke the function.
- **Function time.** The time spent in the function itself.

  `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 `obj.conf` file. Only the `default-bucket` is automatically enabled.

You must enable performance statistics collection and `perfdump`.

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

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

The above examples creates a bucket: `acl-bucket`. To associate this bucket with functions, add `bucket=`*bucket-name* to the `obj.conf` function for which to measure performance.

Example

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

Performance Report

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

The report contains the following information:

- `Average`, `Total`, and `Percent` columns show data for each requested statistic.
- `Request Processing Time` is the total time required by the server to process all requests received.
- `Number of Requests` is the total number of requests for the function.
- `Number of Invocations` is the total number of times that the function was invoked. This number differs from the number of requests because a function can be called multiple times while processing one request. The percentage column for this row is calculated in reference to the total number of invocations for all of the buckets.
- `Latency` is the time in seconds that Proxy Server takes to prepare for calling the function.
- `Function Processing Time` is the time in seconds that Proxy Server spends in 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 example shows performance bucket information in `perfdump`:

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

Total number of requests:               62647125
Request processing time:    0.0343  2147687.2500

default-bucket (Default bucket)
Number of Requests:                     62647125    (100.00%)
Number of Invocations:                3374170785    (100.00%)
Latency:                    0.0008    47998.2500    (  2.23%)
Function Processing Time:   0.0335  2099689.0000    ( 97.77%)
Total Response Time:        0.0343  2147687.2500    (100.00%)
```

Monitoring Current Activity Using the Java ES Monitoring Console

The statistics displayed through the Proxy Server Admin Console is also accessible through the Java ES Monitoring Console. Though the information is the same, it is presented in a different format by using Common Monitoring Data Model (CMM). You can also monitor your server by using the Java ES monitoring tools. For more information about using the Java ES monitoring tools, see Sun Java Enterprise System 5 Monitoring Guide at `http://docs.sun.com/app/docs/doc/819-5081`. Use the same settings to tune the server, irrespective of the monitoring method used.

CHAPTER 2

Tuning Sun Java System Web Proxy Server

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

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

General Tuning Tips

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

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

- Work methodically

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

- Adjust gradually

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

- Start fresh

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

  Read the *Sun Java System Web Proxy Server 4.0.12 Release Notes* and the release notes for your operating system whenever you upgrade your system. The release notes often provide updated information about specific adjustments.

Understanding Threads, Processes, and Connections

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

- "Connection-Handling Overview" on page 28
- "Custom Thread Pools" on page 30
- "Native Thread Pool" on page 31
- "Process Modes" on page 32

Connection-Handling Overview

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

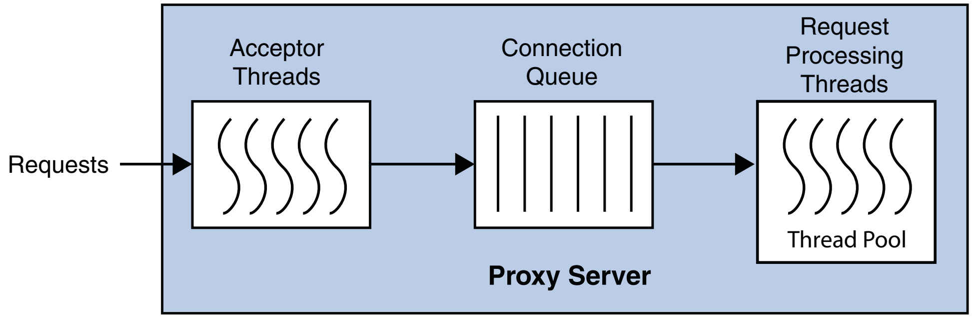

FIGURE 2–1 Proxy Server Connection Handling

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

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

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

The process of adding new session threads is strictly limited by the maximum threads value. For more information on maximum threads, see "Maximum Threads (Maximum Simultaneous Requests)" on page 42.

You can change the settings that affect the number and timeout of threads, processes, and connections in the Admin console.

Low Latency and High Concurrency Modes

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

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

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

Disabled Thread Pools

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

Connection–Handling `magnus.conf` Directives for NSAPI

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

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

For detailed information about these directives, see the *Sun Java System Web Proxy Server 4.0.12 Configuration File Reference*.

Custom Thread Pools

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

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

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

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

In this example, the request is processed as follows:

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

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

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

For more information on defining thread pools, see "thread-pool-init" in *Sun Java System Web Proxy Server 4.0.12 Configuration File Reference*.

Native Thread Pool

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

Proxy Server uses Netscape Portable Runtime (NSPR), which is an underlying portability layer providing access to the host OS services. This layer provides abstractions for threads that are not always the same as those for the OS-provided threads. These non-native threads have lower scheduling overhead, so their use improves performance. However, these threads are sensitive to blocking calls to the OS, such as I/O calls. To make it easier to write NSAPI extensions that can make use of blocking calls, the server keeps a pool of threads that safely support blocking calls. These threads are usually native OS threads. During request processing, any NSAPI function that is not marked as being safe for execution on a non-native thread is scheduled for execution on one of the threads in the native thread pool.

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

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

```
Init funcs="pcheck_uri_clean_fixed_init" shlib="C:/Sun/proxyserver40/lib/custom.dll"
fn="load-modules" NativeThread="no"
```

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

For information on the `load-modules` function, see "load-modules" in *Sun Java System Web Proxy Server 4.0.12 Configuration File Reference*.

Process Modes

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

- "Single-Process Mode" on page 32
- "Multi-Process Mode" on page 33

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

Single-Process Mode

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

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

Multi-Process Mode

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

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

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

When you specify a `MaxProcs` value greater than 1, the server relies on the operating system to distribute connections among multiple server processes (see “`MaxProcs` (UNIX/Linux)” on page 34 for information about the `MaxProcs` directive). However, many modern operating systems do not distribute connections evenly, particularly when there are a small number of concurrent connections.

Because Sun Java System Web Proxy Server cannot guarantee that load is distributed evenly among server processes, you might encounter performance problems if you set Maximum Threads to 1 and `MaxProcs` greater than 1 to accommodate a legacy application that is not thread-safe. The problem is especially pronounced if the legacy application takes a long time to respond to requests, for example, when the legacy application contacts a back-end database. In this scenario, it might be preferable to use the default value for Maximum Threads and serialize access to the legacy application using thread pools. For more information about creating a thread pool, see "thread-pool-init" in *Sun Java System Web Proxy Server 4.0.12 Configuration File Reference*.

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

`MaxProcs` (UNIX/Linux)

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

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

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

Using Monitoring Data to Tune Your Server

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

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

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

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

- "Connection Queue Information" on page 35
- "HTTP Listener (Listen Socket) Information" on page 37
- "Keep-Alive Information" on page 38
- "Thread Information" on page 42
- "File Cache Statistics Information" on page 44
- "Thread Pool Information" on page 48
- "DNS Cache Information" on page 51

Connection Queue Information

In Proxy Server, a connection is first accepted by acceptor threads associated with the HTTP listener. The acceptor threads accept the connection and put it into the connection queue. Then, request processing threads take the connection in the connection queue and process the request. For more information, see "Connection-Handling Overview" on page 28.

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

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

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

The following table shows the information displayed in the Admin Console when accessing monitoring information for the server instance:

TABLE 2–1 Connection Queue Statistics

Present Number of Connections Queued	0
Total Number of Connections Queued	11222922
Average Connections Over Last 1 Minute	90.35
Average Connections Over Last 5 Minutes	89.64
Average Connections Over Last 15 Minutes	54.02

TABLE 2–1 Connection Queue Statistics *(Continued)*

Maximum Queue Size	160032
Peak Queue Size	1853
Number of Connections Overflowed	0
Ticks Spent	5389284274
Total Number of Connections Added	425723

Current /Peak /Limit Queue Length

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

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

  Maximum Queue Size = Thread Pool Queue Size + Maximum Threads + Keep-Alive Queue Size

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

Tuning

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

Total Connections Queued

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

This setting is not tunable.

Average Queue Length

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

This setting is not tunable.

Average Queuing Delay

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

This setting is not tunable.

Ticks Spent

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

This setting is not tunable.

Total Number of Connections Added

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

HTTP Listener (Listen Socket) Information

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

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

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

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

For more information about adding and editing listen sockets, see the *Sun Java System Web Proxy Server 4.0.12 Administration Guide*.

Address

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

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

Tuning

This setting is tunable when you edit an HTTP listener. If you specify an IP address other than `0.0.0.0`, the server makes one less system call per connection. Specify an IP address other than `0.0.0.0` for best possible performance.

Acceptor Threads

Acceptor threads are threads that wait for connections. The threads accept connections and put them in a queue where they are then picked up by worker threads. For more information, see "Connection-Handling Overview" on page 28.

Ideally, you want to have enough acceptor threads so that there is always one available when a user needs one, but few enough so that they do not burden the system. A good rule is to have one acceptor thread per CPU on your system. You can increase this value to about double the number of CPUs if you find indications of TCP/IP listen queue overruns.

Tuning

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

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

Tuning

This setting is tunable when you edit an HTTP listener.

Keep-Alive Information

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

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

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

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

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

TABLE 2–2 Keep-Alive Statistics

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

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

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

Sun Java System Web Proxy Server does not always honor a keep-alive request from a client. The following conditions cause the server to close a connection, even if the client has requested a keep-alive connection:

- The keep alive timeout is set to 0.
- The keep alive maximum connections count is exceeded.
- Dynamic content, such as a CGI, does not have an HTTP `content-length` header set. This applies only to HTTP 1.0 requests. If the request is HTTP 1.1, the server honors keep-alive requests even if the `content-length` is not set. The server can use chunked encoding for these requests if the client can handle them (indicated by the request header `transfer-encoding: chunked`).
- The request is not `HTTP GET` or `HEAD`.
- The request was determined to be bad. For example, if the client sends only headers with no content.

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

Keep-Alive Count

This section in `perfdump` has two numbers:

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

The maximum number of connections allowed in keep-alive mode can be configured using the `MaxKeepAliveConnections` `magnus.conf` directive.

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

Keep-Alive Hits

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

This setting is not tunable.

Keep-Alive Flushes

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

Keep-Alive Refusals

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

Keep-Alive Timeouts

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

Keep-Alive Timeout

The time, measured in seconds, before idle keep-alive connections are closed.

Keep-Alive Poll Interval

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

Keep-Alive Threads

The `KeepAliveThreads` `magnus.conf` directive can be used to specify the number of keep-alive threads.

Tuning for HTTP 1.0-Style Workload

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

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

If users are experiencing connection timeouts from a browser to Proxy Server when the server is heavily loaded, you can increase the size of the HTTP listener backlog queue by setting the HTTP listener's listen queue size to a larger value, such as 8192. The listen queue size can be specified using the "Configure System Preferences" screen in the admin interface.

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

Tuning for HTTP 1.1-Style Workload

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

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

Also, chunked encoding could affect the performance for HTTP 1.1 workload. Tuning the response buffer size can positively affect the performance. A higher response buffer size, set using the `magnus.conf` parameter, ChunkedRequestBufferSize would result in sending a `Content-length:` header, instead of chunking the response.

You can also set the buffer size for a Service-class function in the `obj.conf` file, using the `UseOutputStreamSize` parameter. `UseOutputStreamSize` overrides the value set using the `output-buffer-size` property. If `UseOutputStreamSize` is not set, Proxy Server uses the `output-buffer-size` setting. If the `output-buffer-size` is not set, Web Server uses the `output-buffer-size` default value of 8192.

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

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

Thread Information

Maximum Threads (Maximum Simultaneous Requests)

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

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

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

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

However, in actuality, Internet clients frequently connect to the server and then do not complete their requests. In these cases, the server waits 30 seconds or more for the data before timing out. This wait interval can be configured using the `AcceptTimeout` directive in `magnus.conf`. By setting the default value to less than 30 seconds you can free up threads sooner, but you might also disconnect users with slower connections. Also, some sites perform heavyweight transactions that take minutes to complete. Both of these factors add to the maximum simultaneous requests that are required. If your site is processing many requests that take many seconds, you might need to increase the number of maximum simultaneous requests.

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

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

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

Tuning

You can increase your thread limits in the Admin console by editing the value of "Request Throttle" under "Configure System Preferences".

File Cache Statistics Information

The cache information section provides statistics on how your file cache is being used. The file cache is an in-memory cache that stores frequently accessed objects from the proxy server's disk cache.

For performance reasons, Proxy Server caches as follows:

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

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

```
CacheInfo:
------------------
File Cache Enabled       yes
File Cache Entries       141/1024
File Cache Hit Ratio     652/664 ( 98.19%)
Maximum Age              30
Accelerator Entries      120/1024
Acceleratable Requests   281/328 ( 85.67%)
Acceleratable Responses  131/144 ( 90.97%)
Accelerator Hit Ratio    247/281 ( 87.90%)
```

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

TABLE 2–3 File Cache Statistics

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

TABLE 2–3 File Cache Statistics *(Continued)*

Maximum Heap Cache Size	10735636
Size of Memory Mapped File Content	0
Maximum Memory Mapped File Size	0
Maximum Age of Entries	30

Accelerator Entries

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

Tuning

You can increase the maximum number of accelerator cache entries by increasing the number of file cache entries as described in "File Cache Entries" on page 46. Note that this number will typically be smaller than the File Cache Entries number because the accelerator cache only caches information about files and not directories. If the number is significantly lower than the File Cache Entries number, you can improve the accelerator cache utilization by following the tuning information described in "Acceleratable Requests" on page 45 and "Acceleratable Responses" on page 45.

Acceleratable Requests

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

Tuning

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

Acceleratable Responses

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

Accelerator Hit Ratio

The number of times the response for a request that can be accelerated was found in the accelerator cache.

Tuning

Higher hit ratios result in better performance. To maximize the hit ratio, see the tuning information for "Acceleratable Responses" on page 45.

File Cache Enabled

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

Tuning

The cache is enabled by default. You can disable it in the Admin console at "Configure File Cache" sub-tab in the "Caching" tab.

File Cache Entries

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

Tuning

The available address space for a 32-bit process like the Proxy server is limited to 4Gbytes. The `max-entries` for file cache is based on the number of threads (as specified by `thread-pool/max-threads`), and the connection queue size. It is recommended to cache small, frequently accessed cache files in the file cache and use `perfdump` to ensure that the file cache hit ratio is close to 100%. To achieve this, you may increase file cache size and fine tune the `max-entries` for optimal performance.

File Cache Hit Ratio

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

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

This setting is not tunable.

Maximum Age

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

Maximum Heap Cache Size

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

File Cache Dynamic Control and Monitoring

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

▼ To Control and Monitor the File Cache

1. **Add a `NameTrans` directive to the default object:**

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

2. **Add an `nsfc` object definition:**

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

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

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

   ```
   Sun Java System File Cache Status (pid 3602)

   The file cache is enabled.
   Cache resource utilization

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

   Parameter settings

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

```
BufferSize: 8192 bytes

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

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

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

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

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

Thread Pool Information

If you are using the default settings, threads from the default thread pool process the request. However, you can also create custom thread pools and use them to run custom NSAPI functions. By default, Web Server creates one additional pool, named `NativePool`. In most cases, the native thread pool is only needed on the Windows platform. For more information on thread pools, see "Understanding Threads, Processes, and Connections" on page 28.

Native Thread Pool

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

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

```
Idle/Peak/Limit              1/1/128
Work Queue Length/Peak/Limit  0/0/0
```

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

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

TABLE 2–4 Thread Pools Statistics

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

Idle /Peak /Limit

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

Tuning

You can modify the maximum threads for NativePool by editing the `NativePoolMaxThreads` parameter in `magnus.conf`. For more information, see "`NativePoolMaxThreads` Directive" on page 51.

Work Queue Length /Peak /Limit

These numbers refer to a queue of server requests that are waiting for the use of a native thread from the pool. The `Work Queue Length` is the current number of requests waiting for a native thread, which is represented as Requests Queued in the Admin console.

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

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

Tuning

You can modify the queue size for NativePool by editing the `NativePoolQueueSize` directive in `magnus.conf`. For more information, see "`NativePoolQueueSize` Directive" on page 50.

`NativePoolStackSize` Directive

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

Tuning

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

`NativePoolQueueSize` Directive

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

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

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

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

Tuning

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

`NativePoolMaxThreads` Directive

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

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

Tuning

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

`NativePoolMinThreads` Directive

`NativePoolMinThreads` determine the minimum number of threads in the native (kernel) thread pool.

Tuning

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

DNS Cache Information

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

```
HostDNSCacheInfo:
------------------
enabled             yes
CacheEntries        0/1024
HitRatio            0/0 ( 0.00%)

Async DNS disabled

ClientDNSCacheInfo:
```

```
-----------------
enabled             yes
CacheEntries        0/1024
HitRatio            0/0 ( 0.00%)

Async DNS disabled
```

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

TABLE 2–5 DNS Cache Statistics

Total Cache Hits	62854802
Total Cache Misses	6110
Number of Asynchronous Lookups	0
Lookups in Progress	4
Asynchronous Lookups Enabled	1
Number of Asynchronous Address Lookups Performed	0

Enabled

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

Tuning

By default, the DNS cache is on. You can enable or disable DNS caching in the Admin console at "Configure DNS Caching".

Note: The Proxy server optionally maintains two types of DNS caches. One is a 'Host DNS' cache which caches the results of hostname to ip address lookups done on remote hosts. The second is a 'Client DNS' cache that caches the results of ip address to hostname lookup done on clients.

Cache Entries

This section in `perfdump` shows the number of current cache entries and the maximum number of cache entries. In the Admin Console the current cache entries are shown as Total Cache Hits. A single cache entry represents a single IP address or DNS name lookup. The cache should be as large as the maximum number of clients that access your web site concurrently. Note that setting the cache size too high wastes memory and degrades performance.

Hit Ratio of Cache Hits and Lookups

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

This setting is not tunable.

Async DNS Enabled or Disabled

Async DNS enabled or disabled displays whether the server uses its own asynchronous DNS resolver instead of the operating system's synchronous resolver. By default, Async DNS is disabled. If it is disabled, this section does not appear in `perfdump`.

Tuning the ACL Cache

The Proxy server maintains an ACL Cache that maps between URLs and ACL Lists. The ACL cache improves performance by avoiding the need to build the ACL list applicable to a particular URL for each access.

However, the sheer number of URLs accessed through a Proxy server can cause the ACL Cache to grow to huge sizes. The `magnus.conf` directive called "ACLCacheMax" can be used to restrict the maximum number of entries in the ACL Cache.

Tuning the ACL User Cache (Authentication Cache)

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

This bottleneck can be avoided by increasing the maximum users of the ACL cache at "Configure ACL Cache" under "Preferences" in the Admin console.

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

While it would be extremely unusual to hit this possible performance problem, the number of groups cached in a single ACL cache entry can be tuned with "Proxy Auth Group Cache Size" at "Configure ACL Cache" under "Preferences" in the Admin console.

The maximum age setting of the ACL cache determines the number of seconds before the cache entries expire. Each time an entry in the cache is referenced, its age is calculated and checked against the maximum age setting. The entry is not used if its age is greater than or equal to the maximum age. The default value is 120 seconds. If your LDAP is not likely to change often, use a large number for the maximum age. However, if your LDAP entries change often, use a smaller value. For example, when the value is 120 seconds, the Proxy Server might be out of sync with the LDAP server for as long as two minutes. Depending on your environment, that might or might not be a problem.

Tuning the Proxy Disk Cache to Store Dynamic Content

For a list and description of settings available to fine tune the disk cache for a wider range of responses, including dynamic responses, see “cache-setting” in *Sun Java System Web Proxy Server 4.0.12 Configuration File Reference*. These settings result in a behavior that violates the HTTP standard.

Using Busy Functions

The default busy function returns a "503 Service Unavailable" response and logs a message depending upon the log level setting. You might want to modify this behavior for your application. You can specify your own busy functions for any NSAPI function in the `obj.conf` file by including a service function in the configuration file in this format:

`busy="`*my-busy-function*`"`

For example, you could use this sample service function:

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

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

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

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

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

Two other considerations are footprint and promptness. Footprint is the working size of the JVM process, measured in pages and cache lines. Promptness is the time between when an object becomes dead, and when the memory becomes available.

This is an important consideration for distributed systems. A particular generation size makes a trade-off between these four metrics. For example, a large young generation likely maximizes throughput, but at the cost of footprint and promptness.

CHAPTER 3

Common Performance Problems

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

Note – For platform-specific issues, see Chapter 4, "Platform-Specific Issues and Tips"

`check-acl` Server Application Functions

For optimal server performance, use ACLs only when required.

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

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

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

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

Specific Configurations

Certain performance issues can be tracked down to a specific aspect of the server configuration. For example, it is common practice to use the `assign-name` NameTrans directive in `obj.conf` to apply rules based on request URL pattern. More and more `assign-name` directives get added as time goes by and specific requirements come up, and it is not unusual to see configurations with the number of `assign-name` directives running into hundreds.

Each `assign-name` directive introduces a regular expression comparison, which can be CPU intensive. Hence, as the number of such directives increase, performace start getting affected.

For more information on this specific topic, see Chapter 6, “Scalability Studies.”

Low-Memory Situations

If Proxy Server must run in low-memory situations, reduce the thread limit to a bare minimum by lowering the value of the "Request Throttle" at "Configure System Preferences" in the Admin console.

The server automatically selects many server defaults based on the system resources, for optimal performance. However, if the server's chosen defaults are not suited to your configuration, you can override them. .

The File Cache configuration, especially when non-default, can impact memory usage. Similarly, the ACL Cache size should be controlled using the “ACL Cache Tuning” in *Sun Java System Web Proxy Server 4.0.12 Release Notes*, if needed.

Too Few Threads

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

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

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

The best way to tell if your server is being throttled by too few threads is to see if the number of active sessions is close to, or equal to, the maximum number of threads. To do this, see “Thread Information” on page 42.

Cache Not Utilized

If the file cache is not utilized, your server is not performing optimally.

Check your hit ratio using statistics from `perfdump`, the Admin console Monitoring tab. The hit ratio is the percentage of times the cache was used with all hits to your server. For more information, see "File Cache Statistics Information" on page 44.

Keep-Alive Connections Flushed

A Proxy site that can service 75 requests per second without keep-alive connections might be able to do 200-300 requests per second when keep-alive is enabled. Therefore, as a client requests various items from a single page, it is important that keep-alive connections are used effectively. If the `KeepAliveCount` shown in `perfdump` (Total Number of Connections Added, as displayed in the Admin Console) exceeds the keep-alive maximum connections, subsequent keep-alive connections are closed, or "flushed," instead of being honored and kept alive.

Check the `KeepAliveFlushes` and `KeepAliveHits` values using statistics from `perfdump` or the Number of Connections Flushed and Number of Connections Processed under Keep Alive Statistics on the Monitoring Statistics page. For more information, see "Keep-Alive Information" on page 38.

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

To reduce keep-alive flushes, increase the keep-alive maximum connections. You can do this using the "MaxKeepAliveConnections" `magnus.conf` parameter. The default is based on the number of available file descriptors in the system. By raising the keep-alive maximum connections value, you keep more waiting keep-alive connections open.

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

Log File Modes

Keeping the log files on a high-level of verbosity can have a significant impact on performance. Use the Admin console's "Server Status" tab to modify and track logging settings.

Garbage Collection

Garbage collection can affect performance in some cases, especially with very high cache sizes. Garbage Collection is a CPU and Disk intensive activity that, by default, involves iterating through the entire cache structure and deleting enough files to bring down the cache size as per the limits imposed by the caching configuration.

Garbage Collection is *Automatic* by default, which means that the Proxy server has a dedicated "Garbage Collection Thread" which periodically wakes up to inspect the cache and perform Garbage Collection if needed.

Using the admin GUI, Garbage Collection can be changed to *Scheduled* which disables the in-process Garbage Collection thread. This can help improve performance, as *Scheduled* Garbage Collection is achieved using the `cachegc` command line utility and can be configured to run only at specific times on specific days.

CHAPTER 4

Platform-Specific Issues and Tips

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

- "Solaris Platform-Specific Issues" on page 61
- "Solaris File System Tuning" on page 63
- "Solaris Platform-Specific Performance Monitoring" on page 65
- "Tuning Solaris for Performance Benchmarking" on page 66
- "Tuning UltraSPARC T1–Based Systems for Performance Benchmarking" on page 67

Solaris Platform-Specific Issues

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

- "Files Open in a Single Process (File Descriptor Limits)" on page 61
- "Failure to Connect to HTTP Server" on page 62
- "Connection Refused Errors" on page 63
- "Tuning TCP Buffering" on page 63

Files Open in a Single Process (File Descriptor Limits)

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

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

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

Failure to Connect to HTTP Server

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

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

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

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

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

The Proxy Server HTTP listener's listen queue setting and the related Solaris `tcp_conn_req_max_q` and `tcp_conn_req_max_q0` settings are meant to match the throughput of the Proxy 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 Proxy Server.

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

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

Connection Refused Errors

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

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

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

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

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

Tuning TCP Buffering

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

```
set sq_max_size=512
```

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

Solaris File System Tuning

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

- "High File System Page-In Rate" on page 64
- "Reduce File System Housekeeping" on page 64
- "Long Service Times on Busy Disks or Volumes" on page 64

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

High File System Page-In Rate

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

```
set segmap_percent=25
```

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

Reduce File System Housekeeping

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

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

Long Service Times on Busy Disks or Volumes

Proxy Server's responsiveness depends greatly on the performance of the disk subsystem. Use the `iostat` utility to monitor how busy the disks are and how rapidly they complete I/O requests (the `%b` and `svc_t` columns, respectively). Service times are unimportant for disks that are less than about 30% busy, but for busier disks, service times should not exceed about 20 milliseconds. If your busy disks have slower service times, improving disk performance can help Proxy Server performance substantially.

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

Solaris Platform-Specific Performance Monitoring

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

- "Short-Term System Monitoring" on page 65
- "Long-Term System Monitoring" on page 65
- "Intelligent Monitoring" on page 66

The tools described in this section monitor performance from the standpoint of how the system responds to the load that Proxy Server generates. For information about using Proxy Server's own capabilities to track the demands that users place on the Proxy Server itself, see "Monitoring Server Performance" on page 19.

Short-Term System Monitoring

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

- The `iostat -x 60` command reports disk performance statistics at 60-second intervals.

  Watch the `%b` column to see how much of the time each disk is busy. For any disk busy more than 20% of the time, pay attention to the service time as reported in the `svct` column. Other columns report the I/O operation rates, the amount of data transferred, and so on.
- The `vmstat 60` command summarizes virtual memory activity and some CPU statistics at 60-second intervals.

  Monitor the `sr` column to keep track of the page scan rate and take action if it is too high. Note that "too high" is very different for Solaris 8 and 9 than for earlier releases. Watch the `us`, `sy`, and `id` columns to see how heavily the CPUs are being used. Remember that you need to keep plenty of CPU power in reserve to handle sudden bursts of activity. Also keep track of the `r` column to see how many threads are contending for CPU time, if this remains higher than about four times the number of CPUs, reduce the server's concurrency.
- The `mpstat 60` command gives a detailed look at CPU statistics, while the `netstat -i 60` command summarizes network activity.

Long-Term System Monitoring

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

- Edit the file `/etc/init.d/perf` and remove the `#` comment characters from the lines near the end of the file. For Solaris 10, run the following command:

  ```
  svcadm enable system/sar
  ```

- Run the command `crontab -e sys` and remove the `#` comment characters from the lines with the `sa1` and `sa2` commands. You can adjust how often the commands run and at what times of day depending on your site's activity profile. See the `crontab` man page for an explanation of the format of this file.

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

Intelligent Monitoring

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

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

Solaris 10 Platform-Specific Tuning Information

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

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

Tuning Solaris for Performance Benchmarking

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

TABLE 4–1 Tuning Solaris for Performance Benchmarking

Parameter	Scope	Default Value	Tuned Value	Comments
`rlim_fd_max`	`/etc/system`	65536	65536	Process open file descriptors limit; accounts for the expected load (for the associated sockets, files, and pipes if any).
`sq_max_size`	`/etc/system`	2	0	Controls streams driver queue size; setting to 0 makes it infinite so the performance runs are not hit by lack of buffer space. Set on clients too. Note that setting `sq_max_size` to 0 is not optimal for production systems with high network traffic.
`tcp_time_wait_interval`	`ndd /dev/tcp`	240000	60000	Set on clients too.
`tcp_conn_req_max_q`	`ndd /dev/tcp`	128	1024	
`tcp_conn_req_max_q0`	`ndd /dev/tcp`	1024	4096	
`tcp_ip_abort_interval`	`ndd /dev/tcp`	480000	60000	
`tcp_keepalive_interval`	`ndd /dev/tcp`	7200000	900000	For high traffic web sites, lower this value.
`tcp_rexmit_interval_initial`	`ndd /dev/tcp`	3000	3000	If retransmission is greater than 30-40%, increase this value.
`tcp_rexmit_interval_max`	`ndd /dev/tcp`	240000	10000	
`tcp_rexmit_interval_min`	`ndd /dev/tcp`	200	3000	
`tcp_smallest_anon_port`	`ndd /dev/tcp`	32768	1024	Set on clients too.
`tcp_slow_start_initial`	`ndd /dev/tcp`	1	2	Slightly faster transmission of small amounts of data.
`tcp_xmit_hiwat`	`ndd /dev/tcp`	8129	32768	To increase the transmit buffer.
`tcp_recv_hiwat`	`ndd /dev/tcp`	8129	32768	To increase the receive buffer.

Tuning UltraSPARC® T1–Based Systems for Performance Benchmarking

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

Tuning Operating System and TCP Settings

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

TABLE 4–2 Tuning 64-bit systems for performance benchmarking

Parameter	Scope	Default Value	Tuned Value	Comments
`rlim_fd_max`	`/etc/system`	65536	260000	Process open file descriptors limit; accounts for the expected load (for the associated sockets, files, pipes if any).
`hires_tick`	`/etc/system`		1	
`sq_max_size`	`/etc/system`	2	0	Controls streams driver queue size; setting to 0 makes it infinite so the performance runs are not hit by lack of buffer space. Set on clients too. Note that setting `sq_max_size` to 0 is not optimal for production systems with high network traffic.
`ip:ip_squeue_bind`			0	
`ip:ip_squeue_fanout`			1	
`ipge:ipge_taskq_disable`	`/etc/system`		0	
`ipge:ipge_tx_ring_size`	`/etc/system`		2048	
`ipge:ipge_srv_fifo_depth`	`/etc/system`		2048	
`ipge:ipge_bcopy_thresh`	`/etc/system`		384	
`ipge:ipge_dvma_thresh`	`/etc/system`		384	
`ipge:ipge_tx_syncq`	`/etc/system`		1	
`tcp_conn_req_max_q`	`ndd /dev/tcp`	128	3000	
`tcp_conn_req_max_q0`	`ndd /dev/tcp`	1024	3000	
`tcp_max_buf`	`ndd /dev/tcp`		4194304	
`tcp_cwnd_max`	`ndd/dev/tcp`		2097152	
`tcp_xmit_hiwat`	`ndd /dev/tcp`	8129	400000	To increase the transmit buffer.
`tcp_recv_hiwat`	`ndd /dev/tcp`	8129	400000	To increase the receive buffer.

Note that the IPGE driver version is 1.25.25.

Disk Configuration

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

- Write access logs on faster disks or attached storage.
- If running multiple instances, move the logs for each instance onto separate disks as much as possible.
- Enable the disk read or write cache. Note that if you enable write cache on the disk, some writes can be lost if the disk fails.
- Consider mounting the disks with the following options, which can yield better disk performance: `nologging`, `directio`, `noatime`.

Network Configuration

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

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

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

```
$ifconfig ipge0 group proxyserver
$ifconfig ipge1 group proxyserver
```

Proxy Server Start Options

In some cases, performance can be improved by using large page sizes. To start the Proxy Server with 4 Mbytes pages:

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

CHAPTER 5

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, Proxy Server transparently takes advantage of multiple CPUs. In general, the effectiveness of multiple CPUs varies with the operating system and the workload. Dynamic content performance improves as more processors are added to the system. Because static content involves mostly IO, and more primary memory means more caching of the content (assuming the server is tuned to take advantage of the memory), more time is spent in IO rather than CPU activity.

Memory

As a baseline, Proxy Server requires 64 Mbytes RAM. Multiple CPUs require at least 64 Mbytes for each CPU. For example, if you have four CPUs, install at least 256 Mbytes 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 Kbytes for each peak concurrent user.

Drive Space

You need to have enough drive space for your OS, Proxy server installation, and log files. In most cases, 2 Gbytes total is sufficient. Apart from that, you need to factor in the disk space required for Proxy server's disk cache.

Put the OS, swap or paging file, Proxy Server logs, and Disk cache each on separate hard drives. If your log files fill up the log drive, your OS does not suffer. Also, you will be able to tell whether, for example, the OS paging file is causing drive activity.

Your OS vendor can recommend how much swap or paging space to allocate. Based on testing, Proxy Server performs best with swap space equal to RAM, plus enough to map a size that corresponds to the frequently used portion of the disk cache.

Networking

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

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

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

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

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

CHAPTER 6

Scalability Studies

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

This chapter includes the following topics:

Study Goals

The goal of the tests in the study was to shows how well Sun Java System Web Proxy Server 4.0 scales. The tests also helped to determine the configuration and tuning requirements.

Study Conclusion

When tuned, Sun Java System Web Proxy Server 4.0 provides excellent scalability, reliability and performance, particularly when coupled with a network of suitable capacity and hardware whose chip multithreading capabilities take advantage of Proxy Server 4.0's fully threaded model.

Hardware

- Sun SPARC Enterprise T1000 Server
- UltraSparc T1 processor with 8 1GHz cores and support for 32 simultaneous threads
- 8Gbytes RAM
- Solaris 10 operating system
- Four servers used as test machines to generate the load

Network Configuration

- Four load generating servers connected to the T1000 server via a Gigabit ethernet switch in a single subnet
- Single Gigabit ethernet link

Software

Web Proxy Server system configuration:

- 4Gbytes `tmpfs` cache
- Suitably higher values for RqThrottle (320 — 512)
- Keep alive disabled

Web polygraph benchmarking tool, which is a popular freely available benchmarking tool for caching proxies, origin server accelerators, L4/7 switches, content filters, and other web intermediaries, was used to evaluate the performance of Sun Java System Web Proxy sever 4.0.

Content

The studies were conducted with the following content:

- The content size of each object followed an exponential distribution, with an average content size of 13 Kbytes
- All objects were cacheable with a two minute life cycle

Configuration and Tuning

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

`/etc/system` tuning:

```
set rlim_fd_max=500000
set rlim_fd_cur=500000


set sq_max_size=0
set consistent_coloring=2
set autoup=60
set ip:ip_squeue_bind=0
set ip:ip_soft_rings_cnt=0
set ip:ip_squeue_fanout=1
set ip:ip_squeue_enter=3
set ip:ip_squeue_worker_wait=0

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

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

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

TCP/IP tuning:

```
ndd -set /dev/tcp tcp_conn_req_max_q 102400
ndd -set /dev/tcp tcp_conn_req_max_q0 102400
ndd -set /dev/tcp tcp_max_buf 4194304
ndd -set /dev/tcp tcp_cwnd_max 2097152
ndd -set /dev/tcp tcp_recv_hiwat 400000
ndd -set /dev/tcp tcp_xmit_hiwat 400000
```

Network Configuration

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

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

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

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

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

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

Web Proxy Server Tuning

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

TABLE 6–1 Web Proxy Server Tuning Settings

Component	Default	Tuned
Access logging	`enabled`	disabled
Thread pool	`RqThrottle 128`	`RqThrottle 320`
HTTP listener	Non-secure listener on port `8080`	Non-secure listener on port `8080` ListenQ 8192
Keep alive	`enabled`	`disabled`

Cache in Memory

The `tmpfs` filesystem was used to carve a 4Gbytes filesystem out of memory. This `tmpfs` filesystem, which keeps all files in virtual memory, was used for caching purposes.

```
$ mkdir -p /proxycache
$ mount -F tmpfs -o size=5120m swap /proxycache
```

This creates a 5Gbytes filesystem in main memory. Although only 4Gbytes are actively used by the proxy server, a 5Gbytes filesystem provides some spare room.

Performance Tests and Results

The following table contains the performance results for Sun Java System Web Proxy server 4.0 running on Sun SPARC Enterprise T1000 server.

Target Rate	Throughput (Operations / seconds)	Response (ms)	Error	Network Utilization
6000	5999.70	11.02	0%	78%
6900	6906.71	11.10	0%	88%
7500	7503.58	15.65	0.51%	98%
8100	7925.65	293.03	2.15%	100%
9000	7956.88	365.19	11.59%	100%

-The Target Rate column specifies the target rate for clients submitting requests

-The Error column specifies the percentage of total requests that resulted in an error reported by the clients.

Further measurements indicated that the Sun SPARC Enterprise server had approximately 30% CPU idle time during peak loads of the benchmark test. Hence, it follows that the performance can be potentially increased if additional network bandwidth is made available.

References:

http://www.sun.com/blueprints/0607/820-2142.html

Configuration and Performance

Overloading the server `obj.conf` with too many `assign-name` directives can have an adverse effect on performance. Each `assign-name` directive involves a regular expression comparison which can prove CPU intensive.

The following tables contains the performance results with varying number of `assign-name` directives in the server `obj.conf.`

The first set of data is for a server with cache enabled, and the content server present in the local network. Note that the response time is for a single request.

Number of `assign-name` directives in `obj.conf`	Response time in milliseconds
10	1.05
100	1.45
250	1.8
1000	4.3
2000	7.35
4000	13.65
6000	20.0
8000	26.15
10000	32.5

As can be seen from the performance numbers, the response times show a marked increase once the number of `assign-name` directives cross 100.

The following data was obtained with the cache disabled, and the remote server residing in a remote network.

Number of `assign-name` directives in `obj.conf`	Response time in milliseconds
10	238.5
100	239.7
250	240.3
1000	242.2
2000	245.3
4000	252.3
6000	258.2
8000	264.3
10000	271.2

In the above data, a combination of network delay and the absence of a disk cache tend to hide any performance drop due to the computational delay caused by the high number of `assign-name` directives.

Recommendations:

- Do not let the `assign-name` directives run into hundreds
- Those `assign-name` directives that match commonly accessed URLs should appear earlier in the `obj.conf`

Index

F

H

I

J

K

L

M

S

T

U

V

W