# iPlanet Web Server 6.0 Performance Tuning, Sizing, and Scaling Guide

This guide is intended for advanced administrators only. Be cautious when you tune your server. Do not change any values except in exceptional circumstances. Read this guide and other relevant server documentation before making any changes. Always backup your configuration files first.

The following topics are covered in this guide:

- About Server Performance
- Monitoring Current Activity Using the Server Manager
- Monitoring Current Activity Using the perfdump Utility
- Using Statistics to Tune Your Server
- Using Performance Buckets
- Configuring the File Cache
- Tuning the ACL User Cache
- Using Quality of Service
- Using Load Balancing
- Threads, Processes, and Connections
- Unix/Linux Platform-Specific Issues
- Improving Java Performance
- Miscellaneous magnus.conf Directives
- Miscellaneous obj.conf Parameters
- Common Performance Problems

- Tuning Solaris for Performance Benchmarking
- Sizing and Scaling Your Server
- Scalability Studies

# About Server Performance

iPlanet Web Server was designed to meet the needs of the most demanding, high traffic sites in the world. It runs flexibly on both Unix/Linux and Windows NT, and can serve both static and dynamically generated content. iPlanet Web Server can also run in SSL mode, enabling the secure transfer of information.

Your customers' needs may vary significantly. This guide helps you define your server workload and size a system to meet your performance needs. This guide addresses miscellaneous configuration and Unix/Linux platform-specific issues. It also describes the `perfdump` performance utility and tuning parameters that are built into the server.

## Virtual Servers

Virtual servers add another layer to the performance improvement process. Certain settings are tunable for the entire server, while others are based on an individual virtual server. You can also use the quality of service (QOS) features to set resource utilization constraints for an individual virtual server or class of virtual servers. For example, you can use the quality of service features to limit the number of connections allowed for a virtual server or class of virtual servers.

## Performance Issues

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

Here are some areas to consider:

- Number of peak concurrent users
- Security requirements

  Encrypting your iPlanet Web Server's data streams with SSL makes an enormous difference to your site's credibility for electronic commerce and other security-conscious applications, but it also can seriously impact your CPU load. SSL always has a significant impact on throughput, so for best performance minimize your use of SSL, or consider using a multi-CPU server to handle it.

- Size of document tree
- Dynamic vs. static content

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

  When running an SSL server on Solaris without the use of Java, performance gains can be achieved by enabling `SmartHeap`. `SmartHeap` can be enabled by uncommenting a few lines in the server's start script.

  `SmartHeap` is not currently compatible with Java on iPlanet Web Server 6.0.

# Monitoring Performance

You can monitor the performance of your server by:

- Monitoring Current Activity Using the Server Manager
- Monitoring Current Activity Using the perfdump Utility
- Using Performance Buckets
- Using Quality of Service
- Using Load Balancing

These tools are explained in more detail in the following sections.

# Monitoring Current Activity Using the Server Manager

iPlanet Web Server lets you monitor many performance statistics through the Server Manager user interface and through `stats-xml`. Once statistics are enabled, you can monitor them in the following areas:

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

## Enabling Statistics

You must enable statistics on you iPlanet Web Server before you will be able to monitor performance. This can be done through the Server Manager or editing the `obj.conf` and `magnus.conf` files.

### Enabling Statistics from the Server Manager

To enable statistics from the user interface, follow these steps:

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

   The Enable Statistics / Profiling page appears.
3. Select Yes to enable.
4. Click OK.
5. Click Apply.
6. Select Apply Changes to restart the server for your changes to take effect.

### Enabling Statistics with stats-xml

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

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

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

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

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

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

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

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

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

The above example shows you can also designate the following:

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

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

# Monitoring Statistics

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

To monitor statistics from the Server Manager, follow these steps:

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

3. Make sure that Statistics / Profiling is enabled.
4. Select the refresh interval from the drop-down list under Monitor Web Server Statistics:
   - 5
   - 10
   - 15

   The refresh interval is the number of seconds between updates of the statistics information displayed.
5. Select the type of web server statistics to display from the drop-down list:
   - Connections
   - DNS
   - Keep-alive
   - Cache
   - Virtual Server
6. Click Submit.

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

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

## Virtual Server Statistics

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

# Monitoring Current Activity Using the perfdump Utility

The perfdump utility is an SAF built into iPlanet Web Server that collects various pieces of performance data from the web server internal statistics and displays them in ASCII text. The perfdump utility allows you to monitor a greater variety of statistics than available through the Server Manager.

## Changes to perfdump in this Release

The biggest change in perfdump for the 6.0 release is that the statistics are now unified. Previously perfdump only monitored a single process. Now the statistics, for example file cache size and number of threads, are multiplied by the number of processes to give you a more accurate view of the server as a whole. The categories in the .perf output have changed as well.

## Installing the perfdump Utility

To install perfdump, you need to make the following modifications in obj.conf:

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

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

2. Add the following to the default object:

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

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

   If you need to enable stats-xml, see “Enabling Statistics,” on page 16.

4. Restart your server software.

5. Access perfdump by entering this URL:

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

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

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

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

# Sample perfdump Output

```
ns-httpd pid: 14480

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

    ListenSocket ls1:
    ------------------------
    Address                  http://0.0.0.0:1890
    Acceptor threads          1
    Default virtual server    test

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

    SessionCreationInfo:
    ------------------------
    Active Sessions           1
    Total Sessions Created    48/512

    CacheInfo:
    ------------------
    enabled             yes
    CacheEntries        5/1024
    Hit Ratio           93/190 ( 48.95%)
    Maximum age         30

    Native pools:
    ---------------------------
    NativePool:
    Idle/Peak/Limit               1/1/128
    Work queue length/Peak/Limit  0/0/0

    Server DNS cache disabled

    Async DNS disabled
```

# Using Statistics to Tune Your Server

This section describes the information available through the `perfdump` utility and discusses how to tune some parameters to improve your server's performance. The default tuning parameters are appropriate for all sites except those with very high volume. The only parameters that large sites may regularly need to change are `RqThrottle`, `MaxKeepAliveConnections`, and `KeepAliveTimeout`, which are tunable from `magnus.conf` and the Server Manager.

The `perfdump` utility monitors statistics in these categories:

- Connection Queue Information
- Listen Socket Information
- Keep-Alive / Persistent Connection Information
- Session Creation Information
- Cache Information
- Thread Pools
- DNS Cache Information
- Asynchronous DNS Lookup (Unix / Linux Only)

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

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

The Server Manager Preferences tab includes many interfaces for setting values for server performance, including:

- The Performance Tuning page
- The File Cache Configuration page
- The Treads page (Unix)
- The Native Threads page (NT)
- The Generic Threads page (NT)
- The Magnus Editor

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

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

# Connection Queue Information

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

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

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

## Current /peak /limit

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

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

In the Server Manager 'limit' is referred to as Maximum Number of Queued Connections.

### *Tuning*

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

You can increase the connection queue size by:

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

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

### Total Connections Queued

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

This setting is not tunable.

### Average Queuing Delay

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

This setting is not tunable.

## Listen Socket Information

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

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

```
ListenSocket ls1:
------------------------
Address                  http://0.0.0.0:1890
Acceptor threads          1
Default virtual server    test
```

### *Tuning*

You can create listen sockets through the Server Manager, and edit much of a listen socket's information. For more information, see "Adding and Editing Listen Sockets," on page 150 of the *Administrator's Guide*.

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

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

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

## Address

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

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

### *Tuning*

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

For more information, see "Adding and Editing Listen Sockets," on page 150 of the *Administrator's Guide*.

## Acceptor Threads

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

### *Tuning*

You can tune this number through the user interface when you edit a listen socket. For more information, see "Adding and Editing Listen Sockets," on page 150 of the *Administrator's Guide*.

## Default Virtual Server

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

### *Tuning*

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

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

- Setting or changing the default virtual server information using the Edit Listen Sockets page on the Preferences Tab of the Server Manger. The settings for the default virtual server are on the Connection Group Settings page that appears when you click Groups.
- Editing the `defaultvs` attribute of the `CONNECTIONGROUP` element in the `server.xml` file. For more information, see the chapter on `server.xml` in the *NSAPI Programmer's Guide*.

# Keep-Alive/Persistent Connection Information

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

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

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

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

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

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

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

- `KeepAliveTimeout` is set to 0.
- `MaxKeepAliveConnections` count is exceeded.

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

## KeepAliveThreads

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

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

## KeepAliveCount

This setting has two numbers:

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

### *Tuning*

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

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

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

## KeepAliveHits

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

This setting is not tunable.

## KeepAliveFlushes

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

This setting is not tunable.

## KeepAliveTimeout

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

### *Tuning*

You can change `KeepAliveTimeout` by:

- Editing the `KeepAliveTimeout` parameter in `magnus.conf`
- Setting or changing the `KeepAliveTimeout` value in the Magnus Editor of the Server Manager
- Entering the value in the HTTP Persistent Connection Timeout field of the Performance Tuning page in the Server Manager

## UseNativePoll

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

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

1. Go to the Server Manager Preferences tab and select the Magus Editor.
2. From the drop-down list choose, Keep-Alive Settings and click Manage.
3. Use the drop-down list to set `UseNativePoll` to on.
4. Click OK.

5. Click Apply.
6. Select Apply Changes to restart the server for your changes to take effect.

# Session Creation Information

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

```
SessionCreationInfo:
------------------------
Active Sessions           1
Total Sessions Created    48/512
```

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

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

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

### *Tuning*

You can increase your thread limits by:

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

# Cache Information

The Cache information section provides statistics on how your file cache is being used. The file cache caches static content so that the server handles requests for static content quickly. For tuning information, see “Configuring the File Cache,” on page 42.

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

```
CacheInfo:
------------------
enabled             yes
CacheEntries        5/1024
Hit Ratio           93/190 ( 48.95%)
Maximum age         30
```

## enabled

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

### *Tuning*

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

- Unselecting it from the File Cache Configuration page under Preferences in the Server Manger
- Editing the `FileCacheEnable` parameter in the `nsfc.conf` file. For more information, see the *NSAPI Programmer’s Guide*.

## CacheEntries

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

### *Tuning*

You can set the maximum number of cached entries by:

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

### Hit Ratio (CacheHits / CacheLookups)

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

This setting is not tunable.

### Maximum age

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

#### *Tuning*

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

- Entering or changing the value in the Maximum Age field of the File Cache Configuration page in the Server Manager
- Editing the `MaxAge` parameter in the `nsfc.conf` file. For more information, see the *NSAPI Programmer's Guide*.

## Thread Pools

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

- Thread Pools (Unix /Linux)
- Native Thread Pools (NT)
- Generic Thread Pools (NT)

### Thread Pools (Unix /Linux only)

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

## Native Thread Pools (NT only)

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

```
Native pools:
----------------------------
NativePool:
Idle/Peak/Limit               1/1/128
Work queue length/Peak/Limit  0/0/0
```

Windows NT uses can edit their native thread pool settings using the Server Manager.

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

If you have written your own NSAPI 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.

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

```
Init funcs="pcheck_uri_clean_fixed_init"
shlib="C:/Netscape/p186244/P186244.dll" fn="load-modules"
NativeThread="no"
```

The `NativeThread` flag affects all functions in the `funcs` list, so if you have more than one function in a library, but only some of them use native threads, use separate `Init` lines.

## Generic Thread Pools (NT only)

On NT, you can set up additional thread pools using the Server Manger. Use thread pools to put a limit on the maximum number of requests answered by a service function at any moment. Additional thread pools are a way to run thread-unsafe 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.

## Idle /Peak /Limit

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

### *Tuning*

You can modify the `NativePoolMaxThreads` by:

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

## Work Queue Length /Peak /Limit

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

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

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

### *Tuning*

You can modify the `NativePoolQueueSize` by:

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

## NativePoolStackSize

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

### *Tuning*

You can modify the `NativePoolStackSize` by:

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

## NativePoolQueueSize

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

Setting The `NativePoolQueueSize` lower than the `RqThrottle` value causes the server to execute a busy function instead of the intended NSAPI function whenever the number of requests waiting for service by pool threads exceeds this value. The default returns a “503 Service Unavailable” response and logs a message if `LogVerbose` is enabled. Setting The `NativePoolQueueSize` higher than `RqThrottle` causes the server to reject connections before a busy function can execute.

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

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

### *Tuning*

You can modify the `NativePoolQueueSize` by:

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

## NativePoolMaxThreads

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

A higher value allows more requests to execute concurrently, but has more overhead due to context switching, so “bigger is not always better.” Typically, you will not need to increase this number, but if you are not saturating your CPU and you are seeing requests queue up, then you should increase this number.

### *Tuning*

You can modify the `NativePoolMaxThreads` by:

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

## NativePoolMinThreads

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

### *Tuning*

You can modify the `NativePoolMinThreads` by:

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

# DNS Cache Information

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

## enabled

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

### *Tuning*

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

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

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

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

## CacheEntries (CurrentCacheEntries / MaxCacheEntries)

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

### *Tuning*

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

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

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

  The default cache size is 1024

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

## HitRatio (CacheHits / CacheLookups)

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

This setting is not tunable.

# Asynchronous DNS Lookup (Unix/Linux Only)

You can configure the server to use Domain Name System (DNS) lookups during normal operation. By default, DNS is not enable. If you enable DNS, the server looks up the host name for a system's IP address. Although DNS lookups can be useful for server administrators when looking at logs, they can seriously impact performance. When the server receives a request from a client, the client's IP address is included in the request. If DNS is enabled, the server must look up the hostname for the IP address for every client making a request. Do not enable DNS lookup for high-volume servers.

In order for asynchronous DNS lookups to work correctly, the DNS resolver must be properly configured. See your operating system documentation for details.

## Enable Asynchronous DNS to Avoid Multiple Thread Serialization

DNS causes multiple threads to be serialized when you use DNS services. If you do not want serialization, enable asynchronous DNS. You can enable it only if you have also enabled DNS. Enabling asynchronous DNS can improve your system's performance if you are using DNS.

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

## Caching DNS Entries

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

## Limit DNS Lookups to Asynchronous

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

### enabled

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

#### *Tuning*

You can enable asynchronous DNS by:

- Adding `AsyncDNS on` in the `magnus.conf` file
- Setting the `AsyncDNS` value to on in the Magnus Editor of the Server Manager
- Selecting Async DNS Enabled from the Performance Tuning page under Preferences in the Server Manger

### NameLookups

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

This setting is not tunable.

### AddrLookups

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

This setting is not tunable.

### LookupsInProgress

The current number of lookups in progress.

This setting is not tunable.

## Busy Functions

The default busy function returns a “503 Service Unavailable” response and logs a message if `LogVerbose` is enabled. You may wish to modify this behavior for your application. You can specify your own busy functions for any NSAPI function in the `obj.conf` file by including a service function in the configuration file in this format:

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

For example, you could use this sample service function:

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

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

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

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

my_init(pblock *pb, Session *, Request *)

{

func_insert("service-toobusy", my_custom_busy_function);

}
```

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

# Using Performance Buckets

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

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

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

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

# Configuration

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

First, you must enable performance measurement as described in "Monitoring Current Activity Using the perfdump Utility," on page 19.

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

```
Init fn="define-perf-bucket" name="acl-bucket" description="ACL bucket"
Init fn="define-perf-bucket" name="file-bucket" description="Non-cached
responses"
Init fn="define-perf-bucket" name="cgi-bucket" description="CGI Stats"
```

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

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

# Performance Report

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

For more information, see "Enabling Statistics," on page 16 and "Using Performance Buckets," on page 39.

The report contains the following information:

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

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

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

Total number of requests:                 474851
Request processing time:    0.0010      485.3198

Default Bucket (default-bucket)
Number of Requests:                          597    (  0.13%)
Number of Invocations:                      9554    (  1.97%)
Latency:                    0.0000        0.1526    (  0.03%)
Function Processing Time:   0.0256      245.0459    ( 50.49%)
Total Response Time:        0.0257      245.1985    ( 50.52%)
```

# Configuring the File Cache

The iPlanet Web Server uses a file cache to serve static information faster. In the previous version of the server, there was also an accelerator cache that routed requests to the file cache, but the accelerator cache is no longer used. The file cache contains information about files and static file content. The file cache also caches information that is used to speed up processing of server-parsed HTML.

The file cache is turned on by default. The file cache settings are contained in a file called `nsfc.conf`. You can use the Server Manager to change the file cache settings.

To configure the file cache, follow these steps:

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

   When you enable Transmit File, the server caches open file descriptors for files in the file cache, rather than the file contents, and `PR_TransmitFile` is used to send the file contents to a client. When Transmit File is enabled, the distinction normally made by the file cache between small, medium, and large files no longer applies, since only the open file descriptor is being cached. By default,

Transmit File is enabled on Windows NT, and not enabled on Unix. On Unix, only enable Transmit File for platforms that have native OS support for `PR_TransmitFile`, which currently includes HP-UX and AIX. It is not recommended for other Unix/Linux platforms.

5. Enter a size for the hash table.

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

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

   By default, this is set to 30.

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

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

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

   By default, this is set to 1024.

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

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

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

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

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

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

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

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

10. Click OK.

11. Click Apply.

12. Select Apply Changes to restart your server.

## Using the nocache Parameter

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

For example:

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

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

# Monitoring the File Cache with the Server Manager

To view the file cache statistics the Server Manager, follow these steps:

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

   If you have not yet enabled statistics, when the Monitor Statistics of a Web Server page appears, click OK.
3. Choose a Refresh Interval.
4. From the drop-down list of statistics to be displayed, choose Cache.
5. Click OK.
6. The cache statistics appear, refreshed every 5-15 seconds, depending upon the refresh interval you chose.

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

# File Cache Dynamic Control and Monitoring

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

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

   ```
   NameTrans fn="assign-name" from="/nsfc" name="nsfc"
   ```
2. Add an `nsfc` object definition:

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

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

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

```
iPlanet Web Server File Cache Status (pid 7960)

The file cache is enabled.

Cache resource utilization

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

Parameter settings

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

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

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

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

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

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

# Tuning the ACL User Cache

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

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

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

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

# ACL User Cache Directives

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

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

## ACLCacheLifetime

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

## ACLUserCacheSize

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

## ACLGroupCacheSize

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

## Verifying ACL User Cache Settings

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

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

### *Tuning*

You can turn `LogVerbose` on by:

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

**CAUTION** Do not turn on `LogVerbose` on a production server, because doing so degrades performance and increases the size of your error logs considerably.

# Using Quality of Service

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

For more information, see “Using Quality of Service” on page 213 of the *Administrator’s Guide*.

# Using Load Balancing

Load balancing is dividing the amount of server traffic between two or more computers so that more work gets done in the same amount of time and all online users will generally be served faster.

You can use a third party plug-in, for example the “Resonate Command Module for iPlanet Web Server”, to provide load balancing capabilities. Other companies may also provide load balancing solutions that work with iPlanet Web Server. For more information, contact the load balancing plug-in provider.

# Using libresonate

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

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

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

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

## Library configuration

In order to enable the plug-in, you need to modify `magnus.conf` manually. This should look something like this:

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

Init fn="init-resonate" ThreadPool="sleep"
EventExePath="/tools/ns/bin/perl5" LateInit="yes"
CmdLow="/usr/netscape/ent41/plugins/loadbal/CmdLow.pl"
CmdHigh="/usr/netscape/ent41/plugins/loadbal/CmdHigh.pl"
```

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

**Table 0-1** init-resonate Parameters

| Parameter | Description |
|---|---|
| `ThreadPool` | the name of the thread pool to monitor |
| `Aggressive` | if set to TRUE this argument causes the plug-in use the pool thread count rather than the queue thread count |

**Table 0-1** init-resonate Parameters

| Parameter | Description |
|---|---|
| PollTime | how frequently to check the thread status, by default 2000 milliseconds |
| HighThreshold | defines the queue size/# of threads where HighCmd is executed in order to increase load on the server. The default is 4096. |
| LowThreshold | defines the queue size/# of threads where the LowCmd is executed in order to decrease load on the server. The default is 1. |
| EventExePath | pointer to the script program you want to run (i.e. /usr/bin/perl or /bin/sh). Defaults to perl or perl.exe depending on platform. |
| CmdLow | pointer to the script to be run when the LowThreshold is met |
| ArgsLow | arguments to send to CmdLow |
| CmdHigh | pointer to the script to be run when the HighThreshold is met |
| ArgsHigh | arguments to send to CmdHigh |

**NOTE** You must specify LateInit="yes" when loading this module. This is because the module creates a monitoring thread and this monitoring thread needs to start after ns-httpd has started.

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

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

```
[12/Jun/2000:09:36:35] verbose (20685): Resonate plugin watching
thread pool sleep
[12/Jun/2000:09:36:35] verbose (20685): Resonate plugin aggressive
setting is FALSE
[12/Jun/2000:09:36:35] verbose (20685): Resonate plugin poll time
set to 2000
[12/Jun/2000:09:36:35] verbose (20685): Resonate plugin
HighThreshold set to 5
[12/Jun/2000:09:36:35] verbose (20685): Resonate plugin LowThreshold
set to 1
```

```
[12/Jun/2000:09:36:35] verbose (20685): Resonate plugin event
executable path set to /tools/ns/bin/perl5
[12/Jun/2000:09:36:35] verbose (20685): Resonate plugin low command
set to /usr/netscape/ent41/plugins/loadbal/CmdLow.pl
[12/Jun/2000:09:36:35] verbose (20685): Resonate plugin high command
set to /usr/netscape/ent41/plugins/loadbal/CmdHigh.pl
```

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

```
[12/Jun/2000:09:40:12] verbose (20699): Resonate plugin reducing
load.
[12/Jun/2000:09:40:14] verbose (20699): Resonate plugin reducing
load.
[12/Jun/2000:09:40:16] verbose (20699): Resonate plugin reducing
load.
[12/Jun/2000:09:40:18] verbose (20699): Resonate plugin reducing
load.
[12/Jun/2000:09:40:20] verbose (20699): Resonate plugin reducing
load.
[12/Jun/2000:09:40:30] verbose (20699): Resonate plugin increasing
load.
```

## Testing

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

To configure the sample program, follow these steps:

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

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

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

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

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

   ```
   Init fn="load-modules" funcs="dosleep"
   shlib="/usr/netscape/ent41/plugins/nsapi/examples/dosleep.so"
   pool="sleep"
   ```

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

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

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

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

5. Restart your server.

You should now be ready to test the load balancer plug-in. The NSAPI plug-in will keep the threads busy long enough to simulate whatever load you want. The load-balancing plug-in is tested by retrieving the `.sleep` file you created earlier.

## Sample

Below is a sample `dosleep.c`:

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

#include "nsapi.h"

#define BUFFER_SIZE 1024

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

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

        return REQ_ABORTED;
    }

    duration = atoi(dur);

    /* We need to get rid of the internal content type. */
```

```
    param_free(pblock_remove("content-type", rq->srvhdrs));
    pblock_nvinsert("content-type", "text/html", rq->srvhdrs);

    protocol_status(sn, rq, PROTOCOL_OK, NULL);

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

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

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

# Threads, Processes, and Connections

In iPlanet Web Server 6.0, acceptor threads on a listen socket accept connections and put them onto a connection queue. Session threads then pick up connections from the queue and service the requests. The session threads post more session threads if required at the end of the request. The policy for adding new threads is based on the connection queue state:

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

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

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

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

For more information about these directives, see the *NSAPI Programmer's Guide*.

# Listen Socket Acceptor Threads

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

### *Tuning*

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

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

# Process Modes

You can run your iPlanet Web Server in one of the following two modes:

- iPlanet Web Server with a single process
- iPlanet Web Server with multiple processes

## Single Process Mode

In the single-process mode the server receives requests from web clients to a single process. Inside the single server process many threads are running that are waiting for new requests to arrive. When a request arrives, it is handled by the thread receiving the request. Because the server is multi-threaded, all NSAPI extensions written to the server must be thread-safe. This means that if the NSAPI extension uses a global resource, like a shared reference to a file or global variable, then the use of that resource must be synchronized, so that only one thread accesses it at a time. All plug-ins provided by Netscape/iPlanet are thread-safe and thread-aware, providing good scalability and concurrency. However, your legacy applications may be single-threaded. When the server runs the application, it can only execute one at a time. This leads to server performance problems when put under load. Unfortunately, in the single-process design, there is no real workaround.

## Multi-Process Mode

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

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

In the multi-process mode, the server spawns multiple server processes at startup. Each process contains one or more threads (depending on the configuration) which 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 which requires shared state, it has to synchronize that state across multiple processes. NSAPI provides no helper functions for implementing cross-process synchronization.

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 which is not scalable in a threaded environment, you should use a few processes and many threads, for example, 4 or 8 processes and 128 or 512 threads per process.

## MaxProcs (Unix/Linux)

Use this directive to set your Unix/Linux server in multi-process mode, which may allow for higher scalability on multi-processor machines. If you set the value to less than 1, it will be ignored and the default value of 1 will be used.

### *Tuning*

You can set the value for `MaxProcs` by:

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

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

# Maximum Simultaneous Requests

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

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

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

However, in actuality, Internet clients frequently connect to the server and then do not complete their requests. In these cases, the server waits 30 seconds or more for the data before timing out. You can define this timeout period using the `IOTimeout` directive in `magnus.conf`. The default value is 30 seconds. Also, some sites do heavyweight transactions that take minutes to complete. Both of these factors add to the maximum simultaneous requests that are required. If your site is processing many requests that take many seconds, you may need to increase the number of maximum simultaneous requests. For more information on IOTimeout, see “IOTimeout Information,” on page 62.

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

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

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

### *Tuning*

You can tune the number of simultaneous requests by:

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

# Unix/Linux Platform-Specific Issues

The various Unix / Linux platforms all have limits on the number of files that can be open in a single process at one time. For busy sites, increase that number to 8192.

- Solaris: in `/etc/system`, set `rlim_fd_max`, and `reboot`.
- AIX: run `smit` and check the kernel tuning parameters.
- HP-UX: run `sam` and check the kernel tuning parameters.

These Unix platforms have proprietary sites for additional information about tuning their systems for web servers:

- AIX - `http://www.rs6000.ibm.com/resource/technology/sizing.html`
- IRIX - `http://www.sgi.com/tech/`
- Compaq Tru64 Unix - `http://www.compaq.com/alphaserver/`
- SUN - `http://www.sun.com/sun-on-net/performance/book2ref.html`

# Improving Java Performance

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

- Configuring the Session Manager
- Using Java Heap Tuning
- Using an Alternate Thread Library
- Using Pre-compiled JSPs
- Configuring Class Reloading

## Configuring the Session Manager

You can configure some attributes of the session manager to improve performance problems.

- If you are exhausting the maximum number of sessions, try increasing the value of `maxSessions` from the default value of 1000.
- If you have relatively short-lived sessions, try decreasing the session timeout (`timeOut`) value from the default value of 30 minutes. You can also reduce the frequency at which the session reaper runs by decreasing `reapInterval` from the default value of once every 10 minutes.
- In multi-process mode both `iWSSessionManager` and `MMapSessionManager` use cross-process locks to ensure session data integrity. These can be configured to improve performance as described below.

### Tuning maxLocks (Unix/Linux)

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

### Tuning MMapSessionManager (Unix/Linux)

The following example describes the effect on process size when configuring the `MMapSessionManager`:

```
maxSessions = 1000

maxValuesPerSession = 10

maxValueSize = 4096
```

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

For more information, see the *Programmer's Guide to Servlets*.

## Using Java Heap Tuning

Java heap tuning is application dependent and can be done using the `jvm12.conf` configuration file in the server instance's `config` directory. For additional Java heap and Garbage Collection tuning refer to:

`http://java.sun.com/docs/hotspot/gc`

Though it applies to JDK 1.3.1, concepts are similar and will work for JDK1.2.2_07 which is bundled with iPlanet Web Server 6.0.

## Using an Alternate Thread Library

On Solaris 8 and above, using an alternate thread library, such as `libthread` or `/usr/lib/lwp`, gives optimal performance. You can enable this using the `LD_LIBRARY_PATH` environmental variable in the `start` script.

## Using Pre-compiled JSPs

Compiling JSPs is a resource intensive and relatively time-consuming process. You will improve performance if you pre-compile your JSPs before installing them into your server. More information on compiling JSPs for iPlanet Web Server using the command-line compiler can be found under "Using Java Server Pages" in the *Programmer's Guide to Servlets*.

## Configuring Class Reloading

The `reload-interval` of the `class-loader` element controls the frequency at which the server checks for changes in servlets and JSPs. In a production environment where changes are made in a scheduled manner, set this value to a high number to prevent the server from constantly checking for updates. The default value is 30 seconds. For more information, see the *Programmer's Guide to Servlets.*

# Miscellaneous magnus.conf Directives

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

- IOTimeout Information
- CGIStub Processes (Unix/Linux)
- Buffer Size
- Strict HTTP Header Checking

## IOTimeout Information

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

### Tuning

You can set the`IOTimeout` by:

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

# CGIStub Processes (Unix/Linux)

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

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

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

- MinCGIStubs
- MaxCGIStubs
- CGIStubIdleTimeout
- CGIExpirationTimeout

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

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

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

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

### Tuning

You can set the all of the `CGIStub` processes by:

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

## Buffer Size

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

### Tuning

You can set the buffer size by:

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

## Strict HTTP Header Checking

The server provides strict HTTP header checking, rejecting connections that include inappropriately duplicated headers.

### Tuning

You can suppress this check by setting the `StrictHttpHeaders` directive to `off` in `magnus.conf`:

```
StrictHttpHeaders off
```

# Miscellaneous obj.conf Parameters

You can use some `obj.conf` function parameters to improve your server's performance. In addition to the ones listed below, see "Using the nocache Parameter" on page 44 for information on that parameter.

For more information on using `obj.conf`, see the *NSAPI Programmer's Guide*.

# find-pathinfo-forward

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

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

For example:

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

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

This feature can improve performance for certain URLs by doing fewer stats in the server function `find-pathinfo`. On Windows NT, you can also use this feature to prevent the server from changing "`\`" to "`/`" when using the `PathCheck` server function `find-pathinfo`.

# nostat

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

```
nostat=virtual-path
```

For example:

```
<Object name=default>
NameTrans fn="assign-name" from="/nsfc"  nostat="/nsfc" name="nsfc"
</Object>
<Object name=nsfc>
     Service fn=service-nsfc-dump
     </Object>
```

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

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

# Common Performance Problems

This section discusses a few common web site performance problems to check for:

- Magnus Editor Values
- check-acl Server Application Functions
- Low-Memory Situations
- Under-Throttled Server
- Cache Not Utilized
- Keep-Alive Connections Flushed
- Log File Modes

## Magnus Editor Values

You can set most of the tuning parameter values of the `magnus.conf` file using the Magnus Editor in the Server Manager. However, once you have set the values, the Administration Server does not check if they are valid. Please see the *NSAPI Programmer's Guide* to learn about the default values and acceptable ranges that should be entered in the Magnus Editor fields.

## check-acl Server Application Functions

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

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

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

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

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

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

## Low-Memory Situations

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

## Under-Throttled Server

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

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

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

The best way to tell if your server is being throttled is to see if the number of active sessions is close to, or equal to the maximum number allowed via `RqThrottle`. To do this, see "Maximum Simultaneous Requests," on page 58.

## Cache Not Utilized

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

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

Check your Hit Ratio using statistics from `perfdump` or the Monitor Current Activity page of the Server Manager. The hit ratio is the percentage of times the cache was used with all hits to your server. A good cache hit rate is anything above 50%. Some sites may even achieve 98% or higher.

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

## Keep-Alive Connections Flushed

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

Check the `KeepAliveFlushes` and `KeepAliveHits` values using statistics from `perfdump` or the Monitor Current Activity page of the Server Manager. On a site where keep-alive connections are running well, the ratio of `KeepAliveFlushes` to `KeepAliveHits` is very low. If the ratio is high (greater than 1:1), your site is probably not utilizing keep-alive connections as well as it could.

To reduce keep-alive flushes, increase the `MaxKeepAliveConnections` value in the `magnus.conf` file or the Magnus Editor of the Server Manager. The default value is 200. By raising the value, you keep more waiting keep-alive connections open.

**CAUTION** On Unix/Linux systems, if you increase the `MaxKeepAliveConnections` value too high, the server can run out of open file descriptors. Typically 1024 is the limit for open files on Unix/Linux, so increasing this value above 500 is not recommended.

## Log File Modes

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

# Tuning Solaris for Performance Benchmarking

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

**Table 2** Tuning Solaris for performance benchmarking

| Parameter | Scope | Default Value | Tuned Value | Comments |
|---|---|---|---|---|
| rlim_fd_max | /etc/system | 1024 | 8192 | Process open file descriptors limit; should account for the expected load (for the associated sockets, files, pipes if any). |
| rlim_fd_cur | /etc/system | 64 | 8192 | |
| sq_max_size | /etc/system | 2 | 0 | Controls streams driver queue size; setting to 0 makes it infinity so the performance runs wont be hit by lack of buffer space. Set on clients too. |
| tcp_close_wait_interval | ndd /dev/tcp | 240000 | 60000 | Set on clients too. |
| tcp_time_wait_interval | ndd /dev/tcp | 240000 | 60000 | For Solaris 7 only. Set on clients too. |

**Table 2** Tuning Solaris for performance benchmarking *(Continued)*

| Parameter | Scope | Default Value | Tuned Value | Comments |
|---|---|---|---|---|
| tcp_conn_req_max_q | ndd /dev/tcp | 128 | 1024 | |
| tcp_conn_req_max_q0 | ndd /dev/tcp | 1024 | 4096 | |
| tcp_ip_abort_interval | ndd /dev/tcp | 480000 | 60000 | |
| tcp_keepalive_interval | ndd /dev/tcp | 7200000 | 900000 | For high traffic web sites lower this value. |
| tcp_rexmit_interval_initial | ndd /dev/tcp | 3000 | 3000 | If retransmission is greater than 30-40%, you should increase this value. |
| tcp_rexmit_interval_max | ndd /dev/tcp | 240000 | 10000 | |
| tcp_rexmit_interval_min | ndd /dev/tcp | 200 | 3000 | |
| tcp_smallest_anon_port | ndd /dev/tcp | 32768 | 1024 | Set on clients too. |
| tcp_slow_start_initial | ndd /dev/tcp | 1 | 2 | Slightly faster transmission of small amounts of data. |
| tcp_xmit_hiwat | ndd /dev/tcp | 8129 | 32768 | To increase the transmit buffer. |
| tcp_recv_hiwat | ndd /dev/tcp | 8129 | 32768 | To increase the receive buffer. |

# Sizing and Scaling Your Server

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

- Processors
- Memory
- Drive Space
- Networking

## Processors

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

## Memory

As a baseline, iPlanet Web Server requires 64MB RAM. Multiple CPUs require at least 64MB per CPU. For example, if you have four CPUs, you should install at least 256MB 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 512KB per peak concurrent user.

## Drive Space

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

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

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

## Networking

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

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

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

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

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

# Scalability Studies

This scalability section contains the results of three studies covering the following topics:

- Study Goals
- iWS 6.0 Server Configuration Tested
- Scalability of Dynamic and Static Content
- Scalability Study Settings and Configurations
- iWS 6.0 Scalability Performance and Sizing on F4800
- Performance Results

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

# Study Goals

This study shows how well iPlanet Web Server 6.0 Enterprise Edition scales against 1, 2, 4, and 8 CPUs. This study also helps in determining what kind of configuration (CPU and memory) is required for different types of content. The studies were conducted against the following content:

- 100% Static
- 100% SHTML
- 100% C-CGI
- 100% Perl-CGI
- 100% NSAPI
- 100% Java Servlets
- Java servlet applications

# iWS 6.0 Server Configuration Tested

- Mostly out-of-the box settings
- File Cache configured via `nsfc.conf` for in cache static tests with 40,000 files ranging from 5K to 250K in size
- Tested with two virtual servers (secure and non-secure) on two listen sockets of the same instance
- SSL and non-SSL run without configuring two instances
- Java tests run with both the default and `/usr/lib/lwp` thread libraries
- HTTP 1.0 and HTTP 1.1 for static tests
- `magnus.conf` settings shown below

## Sample magnus.conf Used for iWS 6.0 Sizing

```
#ServerRoot /usr/iplanet/servers
ServerID https-test
ServerName test
ErrorLog /usr/iplanet/servers/https-test/logs/errors
PidLog /usr/iplanet/servers/https-test/logs/pid
User nobody
MtaHost localhost
```

```
DNS off
Security on
ClientLanguage en
AdminLanguage en
DefaultLanguage en
RqThrottle 256
StackSize 131072
CGIWaitPid on
MinCGIStubs 10
MaxCGIStubs 40
TempDir /tmp/https-test

Init fn=flex-init access="$accesslog"
format.access="%Ses->client.ip% - %Req->va
rs.auth-user% [%SYSDATE%] \"%Req->reqpb.clf-request%\"
%Req->srvhdrs.clf-status%
 %Req->srvhdrs.content-length%"
Init fn=load-types mime-types=mime.types
Init fn="load-modules"
shlib="/export/home/iWS6/install/bin/https/lib/libNSServl
etPlugin.so"
funcs="NSServletEarlyInit,NSServletLateInit,NSServletNameTrans,NSSe
rvletService" shlib_flags="(global|now)"
Init fn="NSServletEarlyInit" EarlyInit=yes
Init fn="NSServletLateInit"  LateInit=yes
Init fn="load-modules"
shlib="/export/home/iWS6/install/docs/nsapi/libnsapi-test
.so" funcs="nsapi_test,net_test"
Init fn="load-modules" funcs="shtml_init,shtml_send"
shlib="/export/home/iWS6/in
stall/bin/https/lib/libShtml.so" NativeThread="no"
Init fn="shtml_init" LateInit="yes"
```

# Scalability of Dynamic and Static Content

This section describes a study that tests the scalability of various types of content. They are as follows:

- 100% static
- Dynamic using C-CGI, Perl-CGI, NSAPI, SHTML
- Dynamic content using Java servlets

Java tests were done using both micro benchmarks as well as a shopping cart servlet application.

Near linear scaling was observed for Perl and C-CGI tests using up to 8 CPUs.

SHTML tests also scaled close to 100% up to 4 CPUs. The NSAPI scaling factor is 60%, and its raw performance approached the static content downloads. Static content scalability also peaked to 60%. The testing for some of these cases was limited to 4 CPUs due to network imposed restrictions. Finally, Java applications were found to scale well up to 4 CPUs, with a factor of 70%; and with a factor of 55% beyond 4 CPUs. Java servlet applications showed similar scalability.

Most tests were repeated in SSL enabled mode. SSL static and CGI tests scaled nearly linearly. SSL tests were done with both 100% session cache enabled and session cache totally disabled. Both modes showed uniformly good scalability for all tests. SSL benchmarks for non-Java tests performed optimally with the `SmartHeap` memory allocator. This can be enabled using the `start` script.

# Scalability Study Settings and Configurations

In addition to studying the performance of static and dynamic content of different types, the performance of the keep-alive subsystem on iPlanet Web Server 6.0 was tested and compared against iPlanet Web Server 4.1. The very scalable keep-alive handling solution in iPlanet Web Server 6.0 differs from 4.1 in the following ways:

- The keep-alive connection is not associated with a thread
- Tunable `magnus.conf` parameters for the keep-alive subsystem

  These parameters include:

  - `MaxKeepAliveConnections`
  - `KeepAliveThreads`
  - `KeepAliveTimeout`

## Some Tunings Used for Best Performance

- Static content:
  - File cache (`nsfc.conf`)
  - `MaxKeepAliveConnections` and `KeepAliveTimeout`
  - Pipelined requests
- `KeepAliveThreads` was not changed from the default value

- SSL tests:
  - Enabled `SmartHeap` using the start script
  - Used SSL session cache
- Java tests used the `/usr/lib/lwp` thread library. Set `LD_LIBRARY_PATH` to include `/usr/lib/lwp` in the start script.

## Using the Solaris Network Cache and Accelerator (SNCA)

The Solaris Network Cache and Accelerator (SNCA) is a caching server which provides improved web performance to the Solaris operating environment. It is available on Solaris 8 update 5.

To enable SNCA to work with iPlanet Web Server, follow these steps:

1. In the Solaris operating environment, edit configuration interface file `/etc/nca/nca.if` and add "*" to the first non-comment line.

   This enables SNCA for all network interfaces.

2. Edit `/etc/nca/ncakmod.conf` changing the field 'status' to 'enabled' and changing the field 'ncad_status' to 'enabled'.
3. Edit `/etc/nca/ncalogd.conf` changing the field 'status' to 'enabled'.
4. Reboot the system for changes to take effect.
5. Edit the iPlanet Web Server `server.xml` file so the listen socket on port 80 o includes `family="nca"` as shown below:

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

   The server must be listening on port 80 for this to work.

6. Restart the server for all changes to take effect.

## Setting the TZ Environment Variable on Solaris

In most instances, the timezone environment variable (TZ) is set to the correct value by default when you log on using the local `/etc/default/init` file. You can set the default value if it has not already been set, or you can change the value before starting the server.

To set the TZ environment variable, follow these steps:

1. In `csh` enter `setenv TZ` in the first non-commented line of `/etc/default/init`.

**2.** In Bourne shell enter:

```
TZ = <value>
export TZ;
```

# iWS 6.0 Scalability Performance and Sizing on F4800

- Load generated using E4500 with 12 CPUs
- Server and client connected back-to-back using GB/sec ethernet
- F4800 Server configuration shown below

```
F4800 System Configuration:  Sun Microsystems  sun4u Sun Fire 4800
System clock frequency: 150 MHz
Memory size: 49152 Megabytes
Keyswitch position: On

========================= CPUs ==========================================

             Port  Run    E$   CPU     CPU   Temp   Sensor  Voltage   Sensor
FRU Name      ID   MHz    MB   Impl.   Mask  Deg C  Status  Volts DC  Status
----------  ----  ----  ----  ------  ----  -----  ------  --------  ------
/N0/SB0/P0    0    750   8.0  US-III  3.4   54     Green   1.74      Green
/N0/SB0/P1    1    750   8.0  US-III  3.4   53     Green   1.73      Green
/N0/SB0/P2    2    750   8.0  US-III  3.4   56     Green   1.72      Green
/N0/SB0/P3    3    750   8.0  US-III  3.4   55     Green   1.73      Green
/N0/SB2/P0    8    750   8.0  US-III  3.4   54     Green   1.73      Green
/N0/SB2/P1    9    750   8.0  US-III  3.4   53     Green   1.74      Green
/N0/SB2/P2   10    750   8.0  US-III  3.4   57     Green   1.72      Green
/N0/SB2/P3   11    750   8.0  US-III  3.4   53     Green   1.72      Green
/N0/SB4/P0   16    750   8.0  US-III  3.4   53     Green   1.73      Green
/N0/SB4/P1   17    750   8.0  US-III  3.4   54     Green   1.74      Green
/N0/SB4/P2   18    750   8.0  US-III  3.4   56     Green   1.73      Green
/N0/SB4/P3   19    750   8.0  US-III  3.4   54     Green   1.72      Green

memory 16 way interleaved
```

# Performance Results

For most cases, scalability plots are shown. Performance is shown as a function of the number of CPUs enabled. The following metrics were used to characterize performance:

- Operations per second (ops/sec) = successful transactions per second
- Throughput data transferred in MB or KB per second

- Response time for single transaction (round-trip time) in milliseconds

While operations per second data is shown for all cases, the response time and throughput are shown only where available.

The following sections show the results of the study.

## Static Test

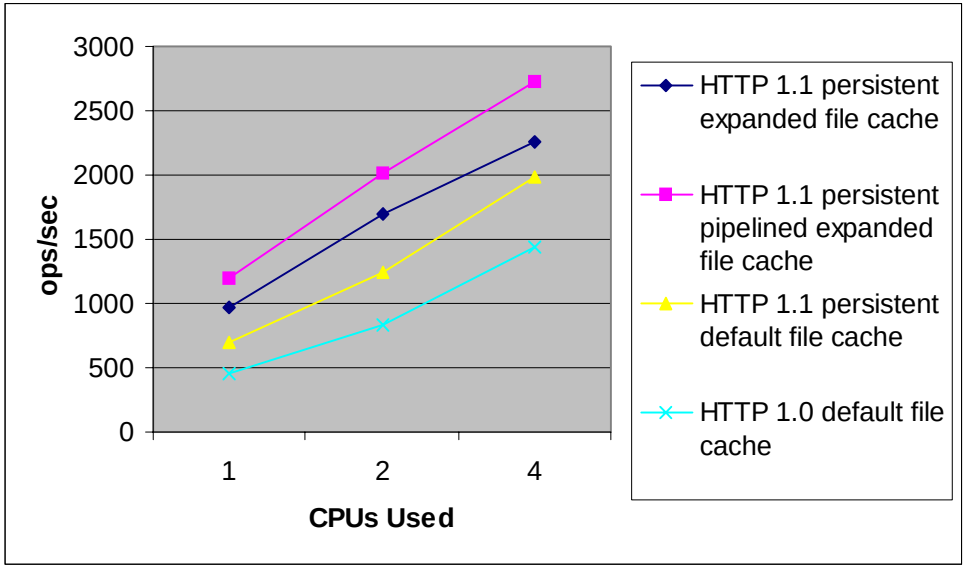
This test was performed with static download of a randomly selected file from a pool of 400 directories each containing 100 files that ranged in size from 5K to 250K. Tests were done with the file cache configured to include all files in the directories. Results are shown for:

- HTTP 1.0 with default file cache
- HTTP 1.1 with default file cache and with persistent connections
- HTTP 1.1 with expanded file cache and persistent connections
- HTTP 1.1 with expanded file cache, persistent connections, and pipelined requests.

Notes:

- 8 CPU data not provided due to single GB ethernet card saturation at 4 CPUs
- 65% scaling achieved

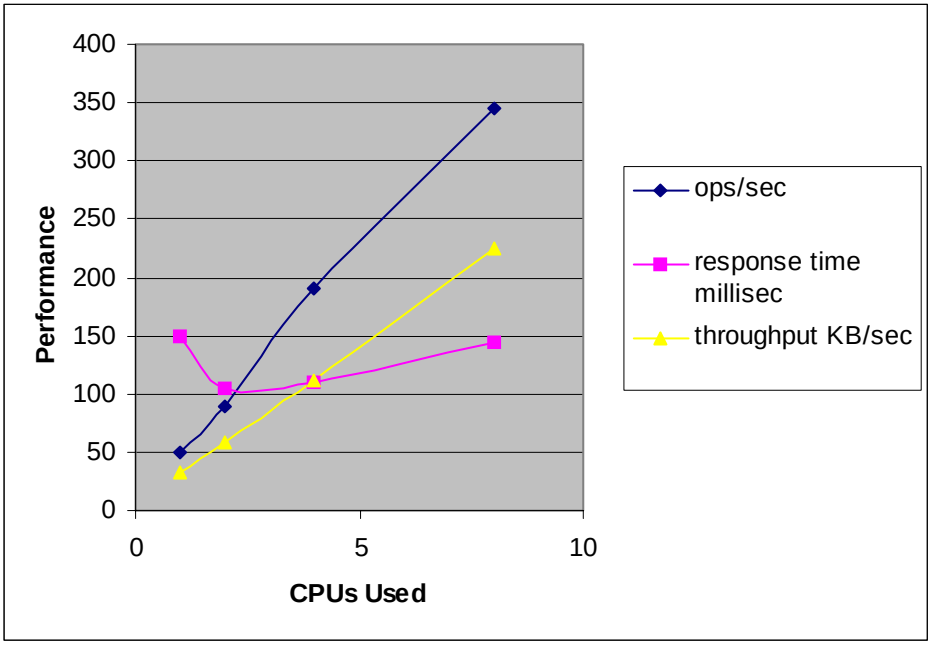
**Figure 1** Static and File Cache Performance



## Perl-CGI (printenv.pl) Test

This test ran against a Perl script called `printenv.pl` that prints the CGI environment. This script outputs approximately 0.5K of data per request. The goal was to saturate the CPUs on the server. The graph below demonstrates that the server scales very well, with a scaling factor close to 100%.

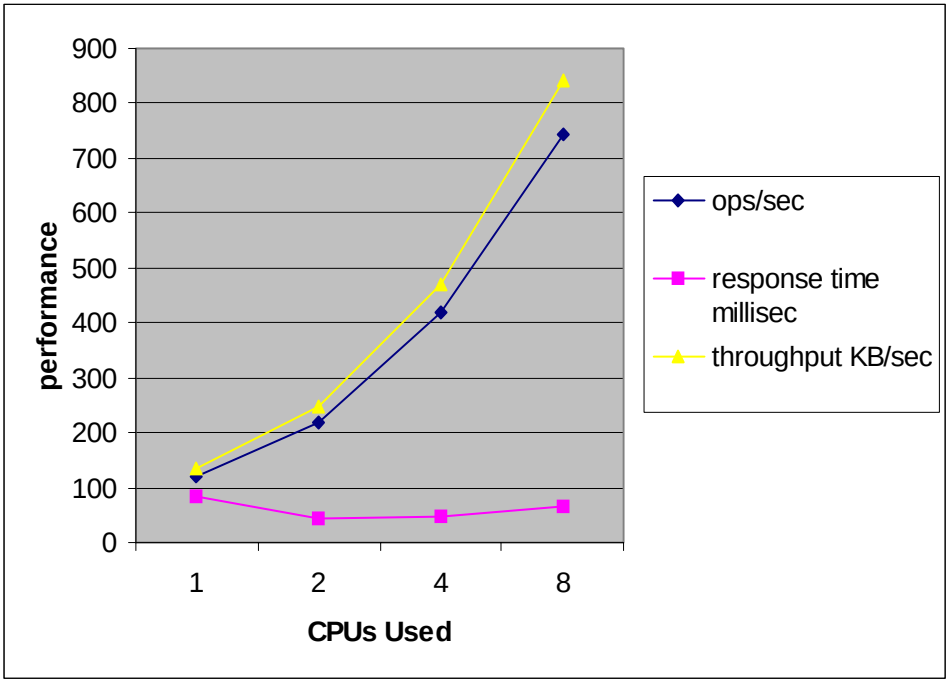
**Figure 2** Perl-CGI Performance



## C-CGI printenv Test

This test was performed by accessing a `C` executable called `printenv`. This executable outputs approximately 0.5K of data per request. The goal was to saturate the CPUs on the server. From the graph below it is clear that the server scales very well for C-CGI content. It scales with a factor of 100%.

**Figure 3** C-CGI printenv Performance



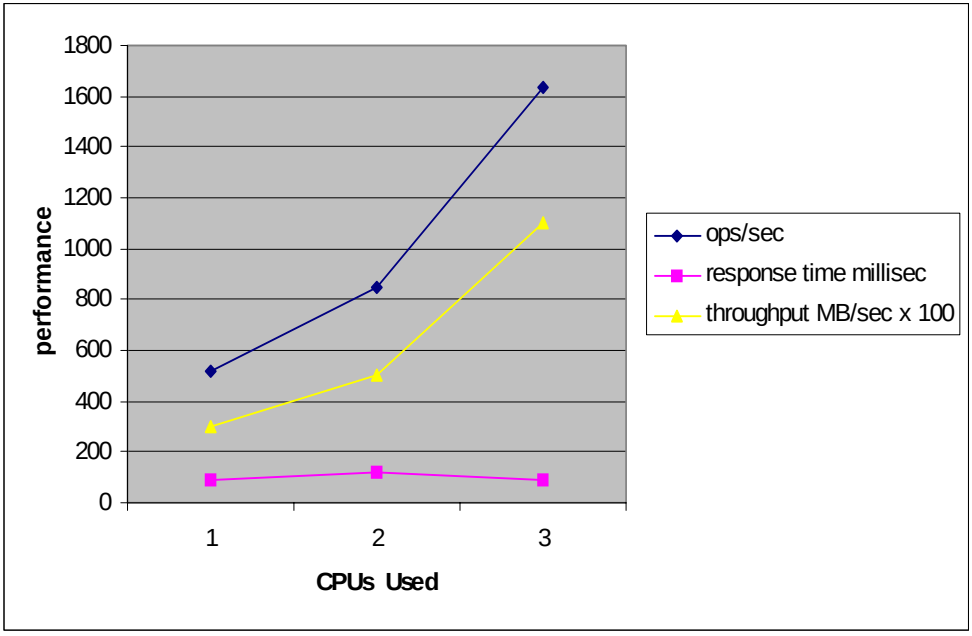
## SHTML Test

This test was run against the file `stresstest.shtml`. This file has three levels of nested includes. The numbers represent average requests per second and average throughput. The goal was to saturate the CPUs on the server. From the graph below it is clear that the server scales well, with a scaling factor of 95%.

Notes:

- Maximum idle CPU time 20% on 4 CPUs
- 8 CPU data unavailable due to network and client limitations
- Throughput plotted as MB per second X 100

**Figure 4** SHTML Performance



## WASP Servlet Test

This test was conducted using the WASP servlet. It prints out the servlet's initialization arguments, environments, request headers, connection/client info, URL information, and remote user information. The goal was to saturate the CPUs on the server. Figure 5 represents average requests per second. From the graph below it is clear that the server scales moderately well for Java servlets content. The scaling factor is 65%.

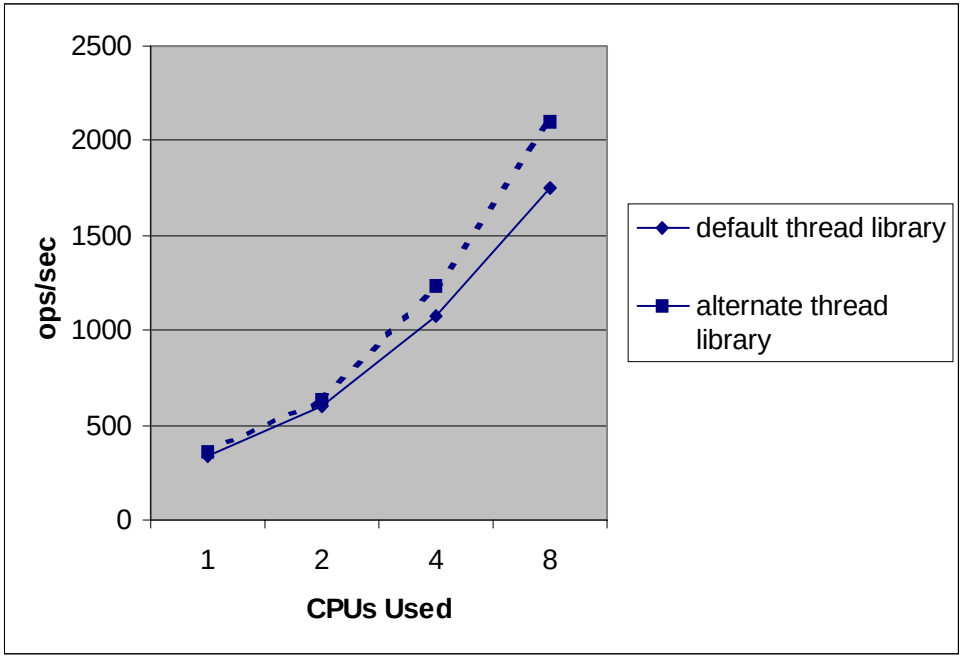
To access the WASP servlet shipped with iWS 6.0, look in the directory where you installed your server for:

`/plugins/servlets/examples/legacy/servlets/WASP/WASPServlet.java`

Notes:

- Servlet deployed as a web application
- Tested with default and alternate (`/usr/lib/lwp`) thread libraries on Solaris 8
- Run in HTTP 1.0 mode without SSL
- Alternate thread library performance up to 20% better for higher CPU counts

**Figure 5** WASP Servlet Performance



## WASP Servlet Throughput and Response Time Test

This test was conducted using the WASP servlet. It prints out the servlet's initialization arguments, environments, request headers, connection / client info, URL information, and remote user information. Figure 6 represent average throughput and response time in seconds. The goal was to saturate the CPUs on the server. From the graph it is clear that the server scales moderately well for Java servlets content. The scaling factor is 65%.

To access the WASP servlet shipped with iWS 6.0, look in the directory where you installed your server for:

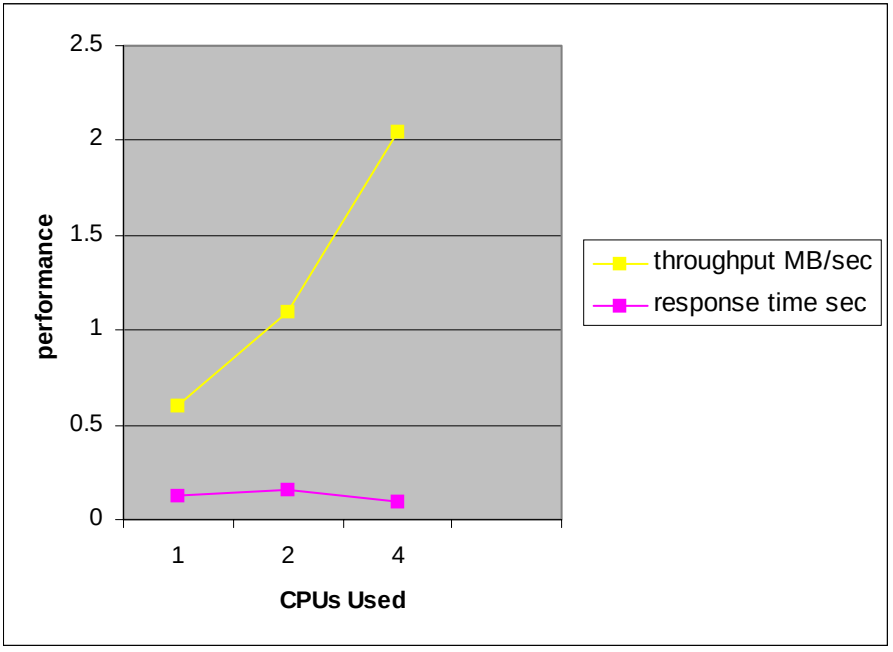
`/plugins/servlets/examples/legacy/servlets/WASP/WASPServlet.java`

- Throughput shown in MB per second response time in seconds

Data provided only for alternate (`/usr/lib/lwp`) thread library on Solaris 8

- 8 CPU performance not shown due to client limitations

**Figure 6** WASP Servlet Throughput and Response Time



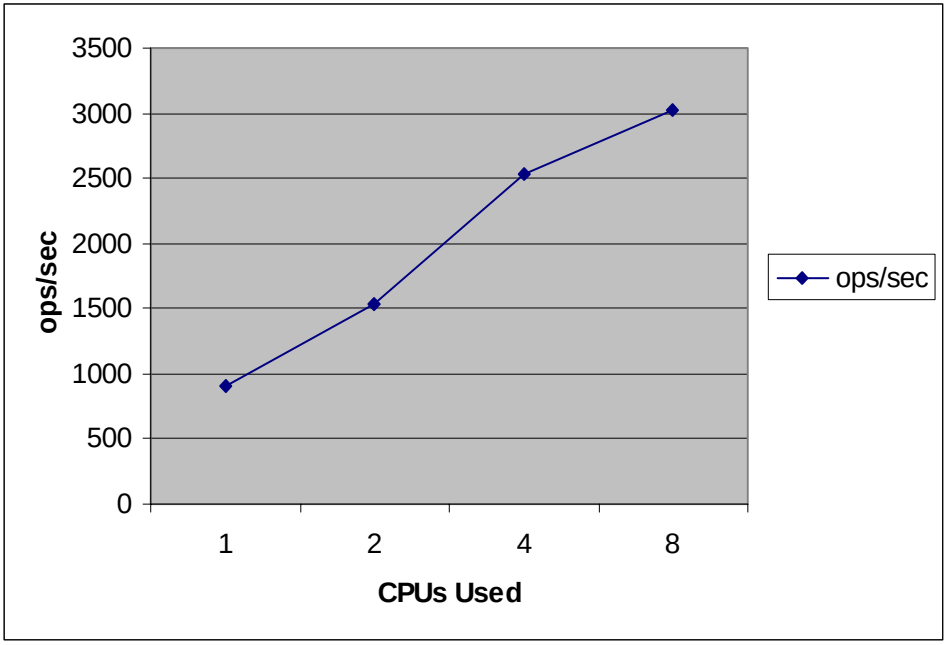
## NSAPI Test

The NSAPI module used in this test was `printenv2.so`. It prints the NSAPI environment variables along with some text to make the entire response 2KB. This graph represents average requests per second. The goal was to saturate the CPUs on the server.

Notes:

- 2KB of data from the server
- 8 CPU data not at full capacity due to client limitations
- Throughput and response time not available due to client limitations

**Figure 7** NSAPI Performance



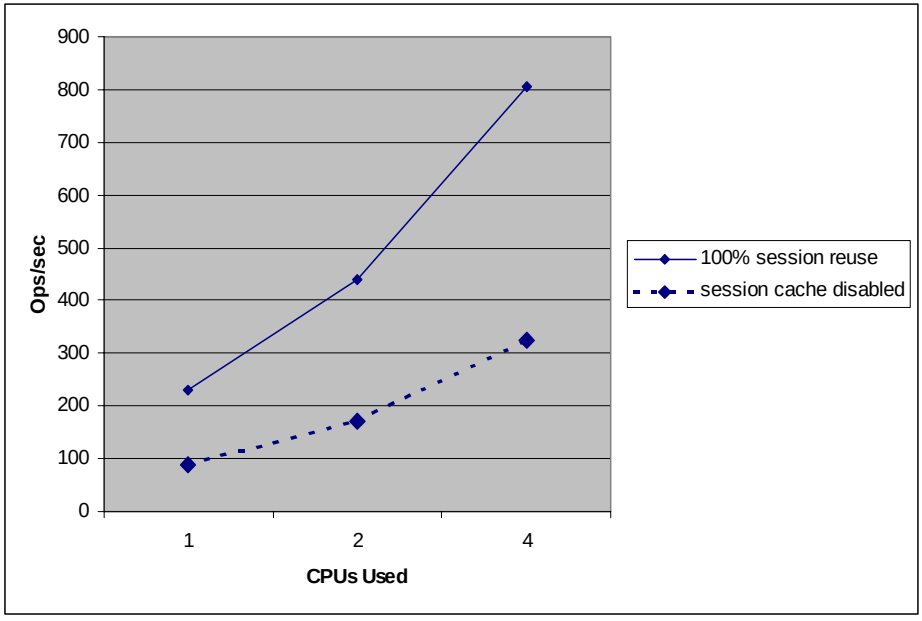
## Static SSL with 100% Session Cache Use Test

A 1KB static SSL file was used for this test. Tests were performed in SSL mode with the SSL session cache both enabled and disabled. The goal was to saturate the CPUs on the server.

Notes:

- `SmartHeap` enabled for this test
- 8 CPU data not available due to client limitations

**Figure 8** Static SSL with 100% Session-Cache Use Performance



## WASP Servlet SSL Test

This test was conducted using the WASP servlet in SSL mode. The goal was to saturate the CPUs on the server. Figure 9 represents average requests per second with SSL session cache enabled and disabled. From the graph below it is clear that the server scales moderately well for Java servlets content. The scaling factor is 75%

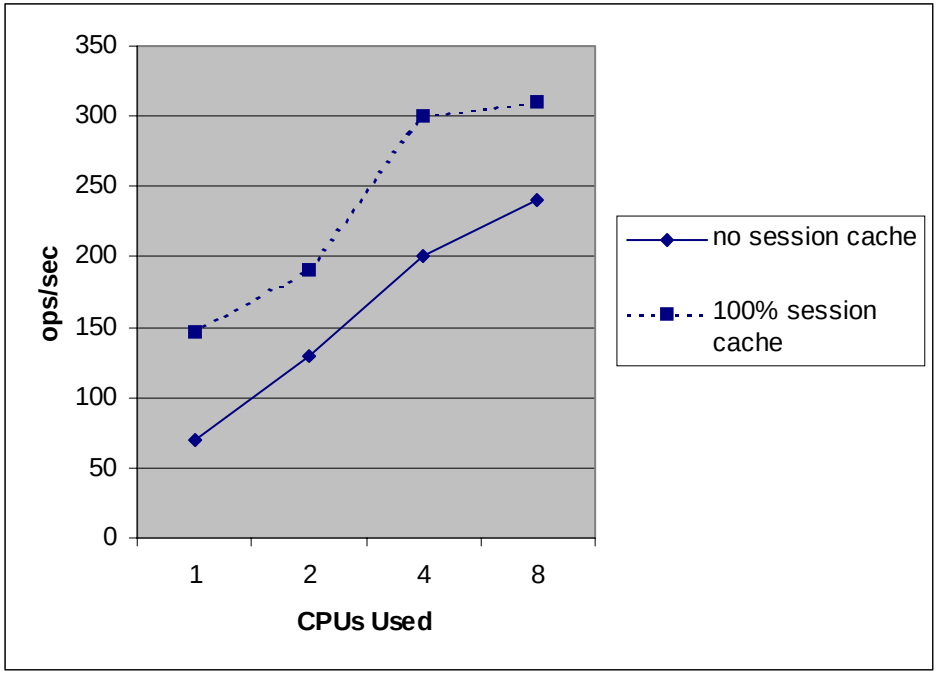
To access the WASP servlet shipped with iWS 6.0, look in the directory where you installed your server for:

`/plugins/servlets/examples/legacy/servlets/WASP/WASPServlet.java`

Notes:

- Alternate (`/usr/lib/lwp`) thread library used for Java servlets
- Throughput data not available due to client limitations
- `SmartHeap` was disabled

**Figure 9** WASP Servlet SSL Performance



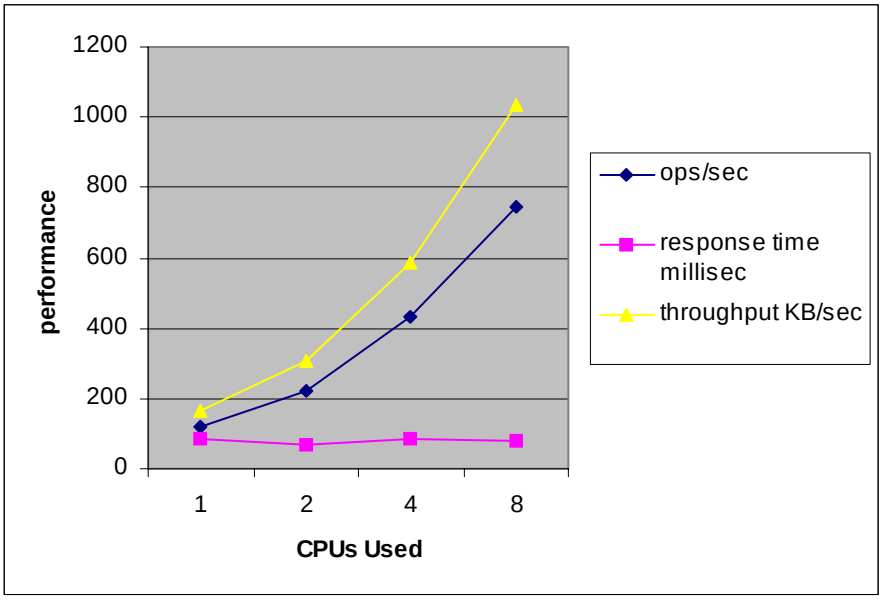
## C-CGI SSL Full Session Cache Test

This test was performed by accessing the `printenv C` executable in SSL mode. Average requests per second and average throughput are shown. The goal was to saturate the CPUs on the server. From the graph it is clear that the server scales very well for C-CGI content with a factor of 100%. SSL session cache was enabled for this test.

Notes:

- SSL cipher rsa_rc4_128_md5

**Figure 10** C-CGI SSL Full Session Cache Performance



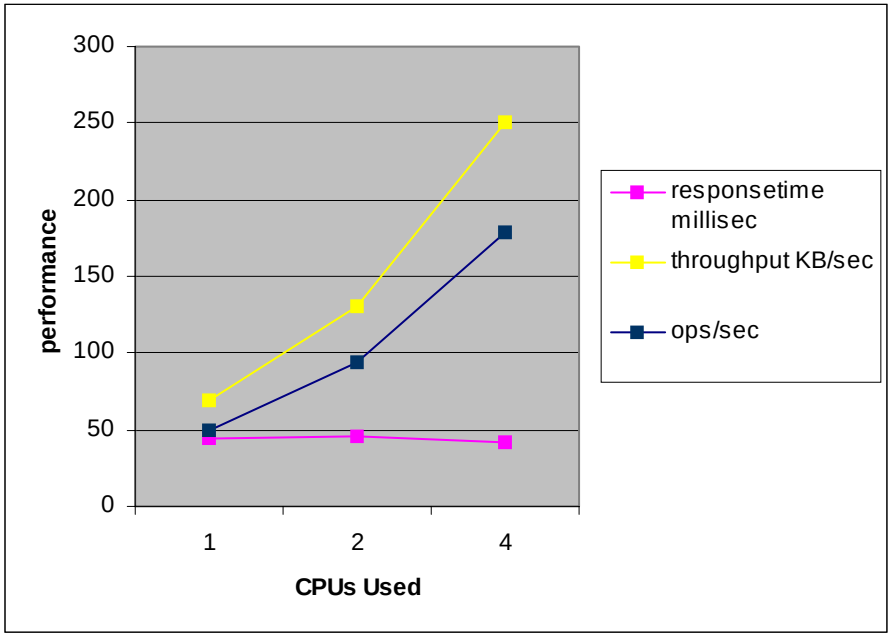
## C-CGI SSL Test

This test was performed by accessing the `printenv C` executable in SSL mode. Average requests per second and average throughput are shown. The goal was to saturate the CPUs on the server. From the graph below it is clear that the server scales very well for C-CGI content with a factor of 100%. SSL session cache was disabled for this test.

Notes:

- SSL ciphers rsa_rc4_128_md5
- 8 CPU data not available due to client limitations

**Figure 11** C-CGI SSL Performance



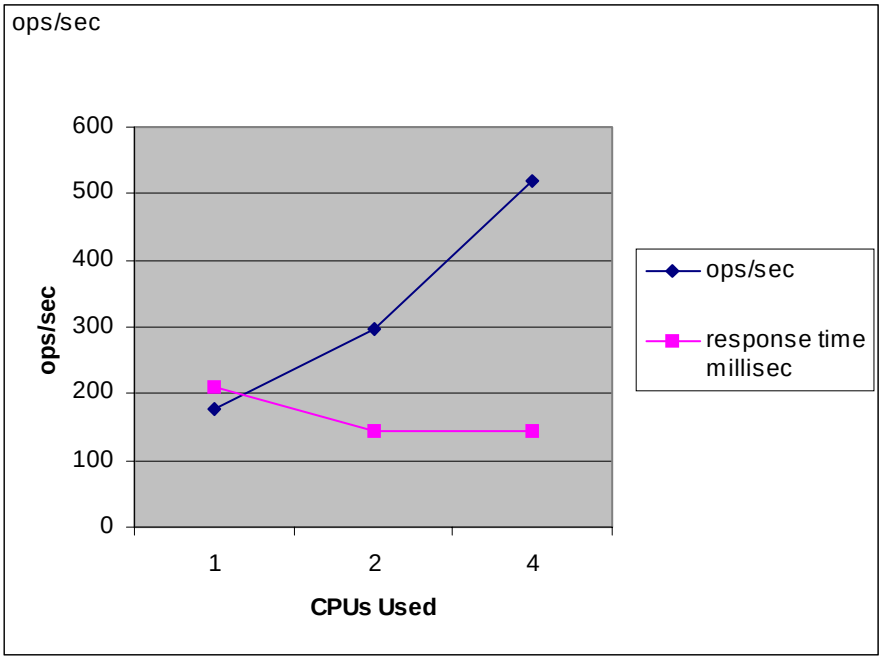
## Java Bookstore Application Test

This Java servlet web application test used multiple servlets and static image files simulating an online bookstore. Each servlet session involved multiple actions sharing a login using userID and password. Other actions included browsing the bookstore index, adding books to the shopping cart, searching by author and subject, and finally checkout and order. Final checkout and order invalidated the session. Each client login represented a session, and several concurrent clients were used during the benchmark. The Oracle database was used, with Oracle JDBC thin driver used for connections. Results below show the number of successful transactions and response time as a function of CPUs used. Close to 75% server scaling was achieved.

Notes:

- Customized connection pools for optimal performance
- Alternate (`/usr/lib/lwp`) thread library used for optimal performance

**Figure 12** Java Bookstore Application Performance



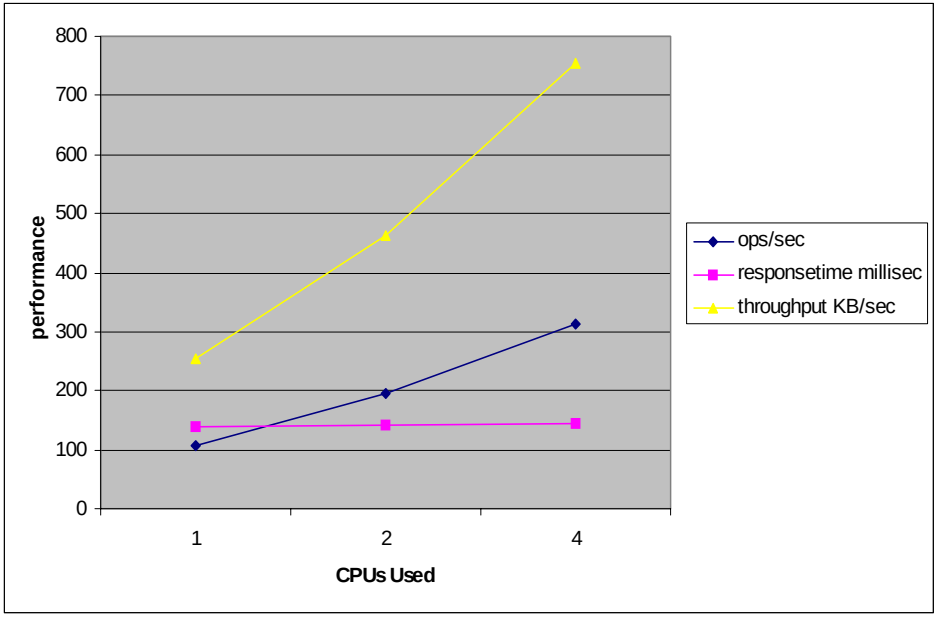
## Java Servlet Bookstore Test

This Java servlet application test used multiple servlets and static image files simulating an online bookstore. Each servlet session involved multiple actions sharing login, database, userID and password. Other actions included browsing the bookstore index, adding books to the shopping cart, searching by author and subject, and final checkout and order. Final checkout and order invalidated the session. Each client login represented a session and the similar shopping session was implemented for all clients. The Oracle database was running on the same system as the server, and the Oracle JDBC thin driver was used for connections. 80% scaling was achieved.

Notes:

- SSL enabled
- Session cache disabled

**Figure 13** Bookstore Performance in Secure Mode



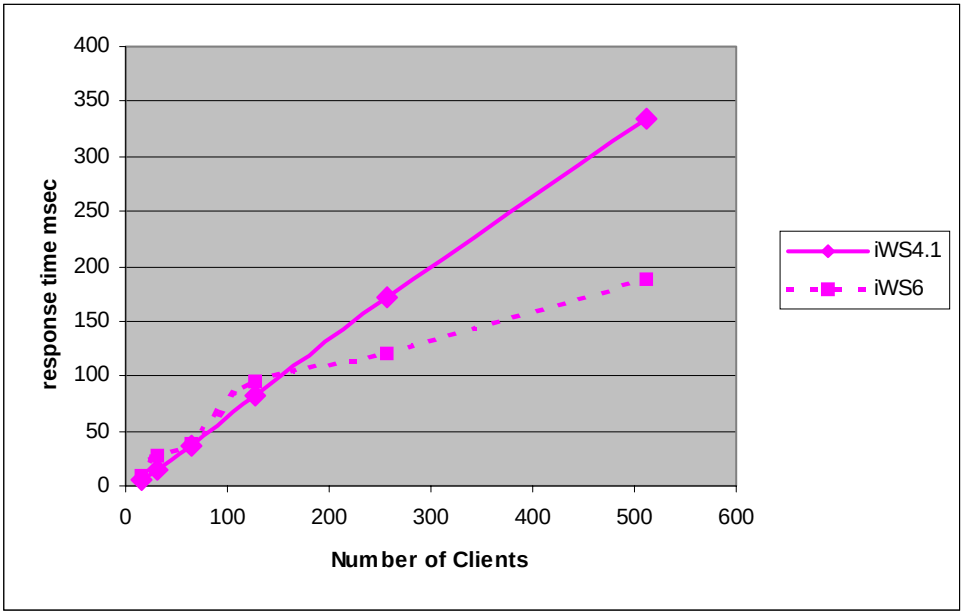
## Keep-Alive Subsystem for Static Content Test

This test was performed with static download of a randomly selected file from a pool of 400 directories each containing 100 files. Results are shown for iWS 4.1, and iWS 6.0 with persistent connections and pipelined requests. Tests were conducted with a file cache configured to include all files in the directories. This graph shows the response time as a function of concurrent keep-alive connections. From this test it is clear that the iWS 6.0 keep-alive subsystem performed nearly 2X better than iWS 4.1, especially at higher client counts.

Notes:

- Tested using a `depth=5` pipelined static download of random file from 40,000 files
- Compared iWS4.1 and iWS 6.0 performance
- `magnus.conf RqThrottle 128`
- Unable to test beyond 512 clients due to network limitations

**Figure 14** Keep-Alive iWS 6.0 Versus iWS 4.1 Performance



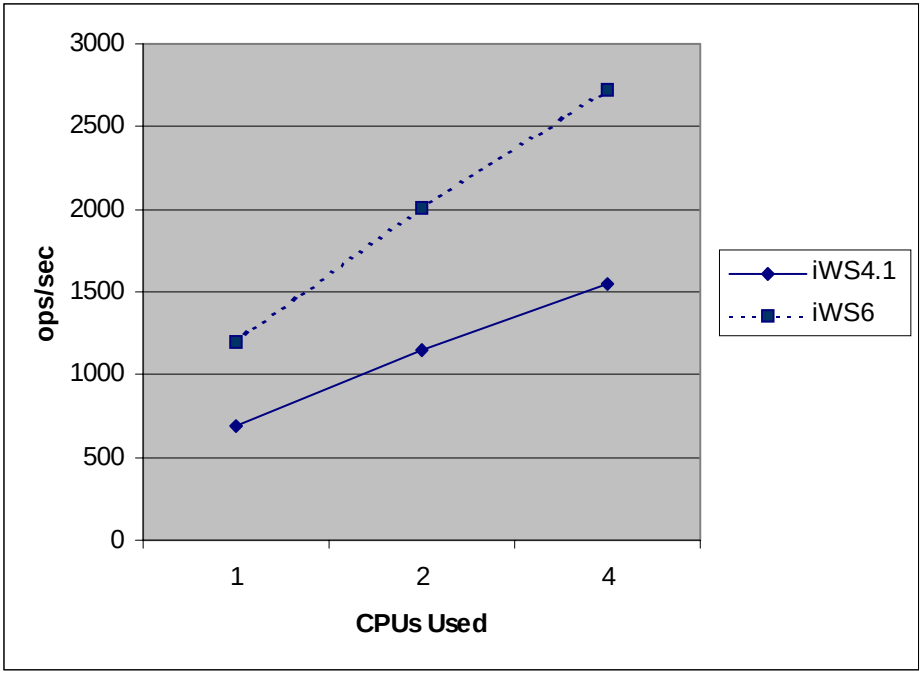
## iWS 6.0 and iWS 4.1 Keep-Alive Performance Comparison

This test was performed with static download of a randomly selected file from a pool of 400 directories each containing 100 files ranging in size from 5 to 250K. Results are shown for iWS 4.1 and iWS 6.0 with persistent connections and pipelined requests. Tests were conducted with a file cache configured to include all files in the directories. The graph below demonstrates iWS 6.0 requests per second performance against iWS 4.1. From this test it is clear that the iWS 6.0 keep-alive subsystem performed nearly two times better than iWS 4.1, especially at higher client counts.

Notes:

- Random pipeline length between 5-15
- Cache configured to contain all 40,000 files
- Keep-Alive settings: `MaxKeepAliveConnections` = 1024, `KeepAliveTimeout` = 3600

**Figure 15** Comparison of iWS 6.0 and iWS 4.1 Keep-Alive Performance



## Java Web Application Stress and Stability Test

This test utilized sessions, request dispatcher, JSPs, and tag libraries.

- Solaris 8 4X400 Mhz UltraSPARC.
- Over 1.8 million servlets and JSPs served per hour
- 76-78% CPU utilization (over 20% idle time)

**Figure 16** Java Web Application Stress and Stability