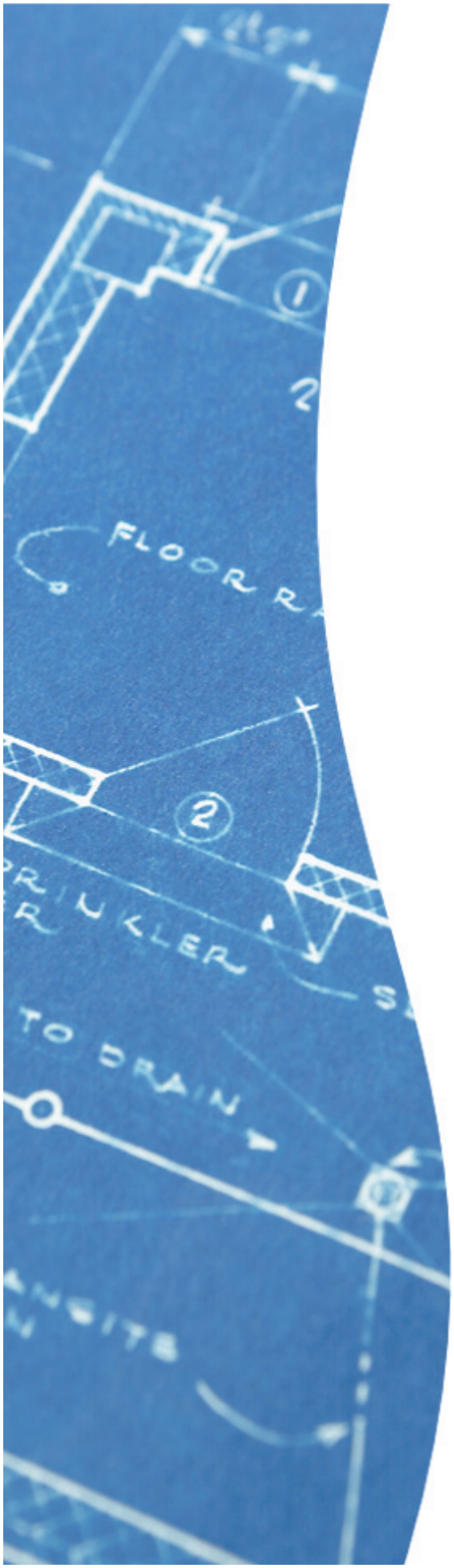

AN OPEN SOURCE WEB SOLUTION

Lighttpd Web Server and Chip Multithreading Technology

Reference Implementation

Amanda Waite, Sun Microsystems

Sun BluePrints™ Online — September 2008

Part No 820-5633-10
Revision 1.0, 9/23/08

Table of Contents

An Open Source Web Solution

With more users interacting, working, purchasing, and communicating over the network than ever before, Web 2.0 infrastructure is taking center stage in many organizations. Demand is rising, and companies are looking for ways to tackle the performance and scalability needs placed on Web infrastructure without raising IT operational expenses. Today companies are turning to efficient, high-performance, open source solutions as a way to decrease acquisition, licensing, and other ongoing costs and stay within budget constraints.

The combination of open source Lighttpd Web server software and Sun servers with CoolThreads™ technology provides a scalable, high-performance, and cost-effective solution for Web environments. This Sun BluePrints™ article describes a reference implementation based on the Lighttpd Web server software and Sun SPARC® Enterprise T5120 servers, and explores its performance and scalability when running dynamic workloads. Workload configuration and testing procedures are described, as well as tuning and optimization steps that can be taken to determine optimal configurations for performance and scalability. High-level throughput and latency characteristics also are presented, and indicate the solution can deliver an average of 416 operations per second for 2,250 users with 90 percent response time for all operations fulfilled within the targets set by the workload definition.

Reference Implementation

The reference implementation consists of a Sun SPARC Enterprise T5120 server running the Solaris™ Operating System (OS) and the Lighttpd Web server software that handles requests from clients (Figure 1). Sun SPARC Enterprise T5120 servers with CoolThreads technology blend the performance and scalability of midrange servers with the economies of energy-efficient, chip multithreading (CMT) designs. Incorporating UltraSPARC® T2 processors, these servers provide up to eight cores and 64 simultaneous execution threads on a single processor to handle rising multithreaded Web workload volumes.

A Sun Fire™ X4200 server provides access to a back-end database. Sun Fire X4200 servers are fast, energy-efficient, and reliable one-way to four-way x64 servers. Designed to deliver increased service levels while also offering lower operational costs and better asset utilization, these two-socket, four-way servers include built-in redundancy and remote management capabilities.

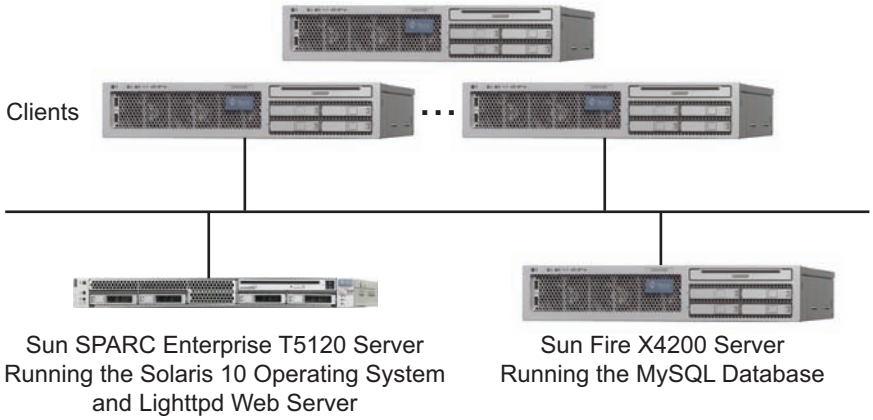

Figure 1. An open source Web solution reference implementation

The Web 2.0 Kit

In order to understand the performance and scalability characteristics of the solution, Sun used the Web 2.0 kit designed to test workloads with a high level of user interactions, such as those utilized in typical Web 2.0 sites.

The Web2.0 kit is a reference architecture for Web 2.0 technologies, and provides a sample Web 2.0 application implemented using three different technology stacks: the Java™ Platform, Enterprise Edition (Java EE), PHP: Hypertext Preprocessor (PHP), and Ruby on Rails (RoR). The Web 2.0 kit can be used in the following ways to:

- Evaluate the differences in the various languages/frameworks for RoR, JEE and PHP (e.g by comparing how a functionality is implemented in each)
- Evaluate the infrastructure technologies for each implementation (e.g. Apache vs lighttpd)
- Compare the performance of the various technologies (e.g PHP vs RoR)

The application in the Web 2.0 Kit implements a social events calendar with features such as Asynchronous JavaScript™ and XML (AJAX), tagging, tag cloud, comments, ratings, feeds, mashups, extensive use of data caching, use of both structured and unstructured data, and a high data read/write ratio that is typical of applications in this space.

The Web 2.0 Kit also includes a workload generator implemented using the Faban harness and driver framework. See *http://faban.sourcenet.net* for more information.

Hardware and Software Configuration

The workload test utilized one Sun SPARC Enterprise T5120 server and four Sun Fire X4200 servers, all running the Solaris 10 OS and the Lighttpd Web server software version 1.4.18 (Table 1). The Sun SPARC Enterprise T5120 server was the system under test, and the Sun Fire X4200 servers were used to run the database and generate request loads (Figure 2). All software utilized was obtained from the Optimized Open Source Software Stack (Cool Stack) available from *http://cooltools/sunsource.net/coolstack*.

Table 1. Hardware and software configuration

Hardware Component	System Type	Configuration Details
System Under Test	Sun SPARC Enterprise T5120 server	8 cores, 64 threads, 1.2 GHz, 32 GB RAM
Loader 1	Sun Fire X4200 server	2 dual-core AMD 2200 processors, 8 GB RAM
Loader 2	Sun Fire X4200 server	2 dual-core AMD 2200 processors, 8 GB RAM
Loader 3	Sun Fire X4200 server	2 dual-core AMD 2200 processors, 8 GB RAM
Database	Sun Fire X4200 server	2 dual-core AMD 2200 processors, 8 GB RAM
Gigabit Ethernet Switch	Linksys SRW2048	48 ports
Software Component	**Software**	**Version**
Operating System	Solaris 10 OS	8/07
Web Server	Lighttpd software	1.4.18 (from Cool Stack version 1.2)
PHP	PHP	5.2.4 (from Cool Stack version 1.2)
Database	MySQL™ database	5.0.45 (from Cool Stack version 1.2)
Test Harness	Faban	Various versions

- *System Under Test (SUT)*
  A Sun SPARC Enterprise T5120 server configured with one 1.2 GHz, 8-core UltraSPARC T2 processor and four Gigabit Ethernet network interface cards (NIC) (`e1000g`) served as the system under test. The primary network interface card, `e1000g0`, was connected to a subnet. The remaining three cards were connected to a switch dedicated to a group of test systems to provide a variety of configuration options. A virtual local area network (VLAN) was created on the switch for the systems involved in the testing effort.

- *Load generating systems (Faban agents)*
  Multiple load generating systems were used to drive sufficient load onto the system under test. Each load generating system consisted of a Sun Fire X4200 server with two primary Gigabit Ethernet network interface cards (`nge`), and two additional Gigabit Ethernet network interface cards (`e1000g`). The primary network interface cards were connected to a subnet, while the additional cards were connected to the same dedicated switch as the system under test.

- *Master system*
  The Faban facility requires a system that acts as a master. The master can be one of the agents used for testing, or a separate system. The majority of the testing effort utilized a master that also acted as an agent. The Faban master coordinates the agents and gathers data in a central repository. One, two, and three agents were used during the testing effort, with agents added as the number of concurrent users increased.

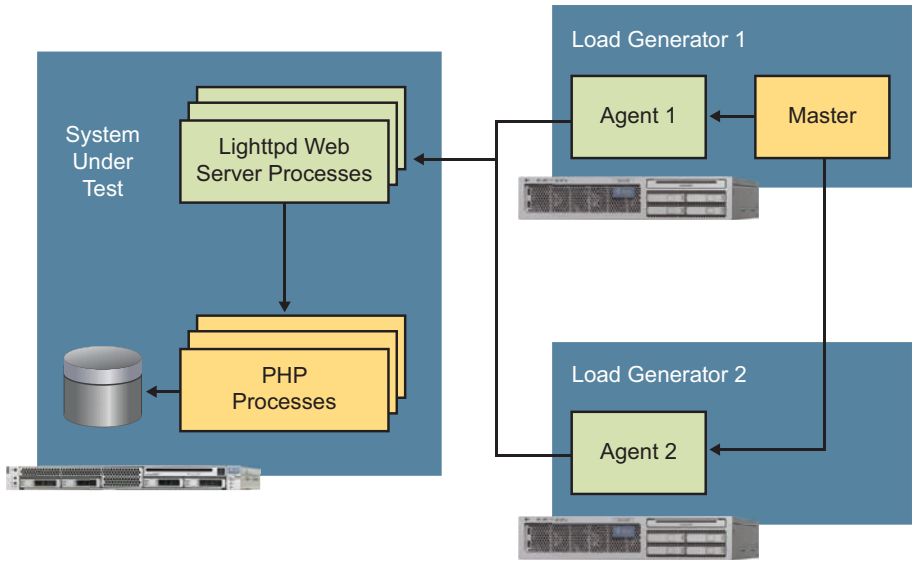

Figure 2. Configuration topology

Workload Test Descriptions

The workload tests conducted utilized the Faban harness, Faban driver framework, and a Web 2.0 kit driver. The Faban driver was designed to drive the application in the Web 2.0 Kit and to simulate real world interactions by a number of concurrent users.

The Faban Harness and Driver Framework

The Faban harness is infrastructure for hosting workloads and automating execution. It gathers data from a test run and presents it to users in graphical form through a Web interface. Faban can manage the systems used in a workload test, including the system under test, load generation systems, and even database and cache servers commonly used in complex workload testing. The combination of the Faban harness and driver framework controls most aspects of a test run, and performs the following tasks:

- Manages multiple Faban agents running on one or more systems.
- Coordinates the agents at startup to help ensure the server is not overloaded. This includes a graduated increase in concurrent users, as well as a ramp up period designed to warm up the application. The goal is to achieve an optimal running state when actual measurements begin to take place.

- Controls the running of statistical tools on the system under test, database server, Memcached servers, and agents, and gathers all data at the end of the test run.
- Works with the Faban driver to control think or cycle times between operations. The workload test described in this document used a negative exponential, resulting in a spread of think times between one and 25 seconds, with an average of five seconds.
- Collects the statistics for each operation, calculates averages, and tests against the targets defined in the driver.
- Presents results in textual and graphical form.
- Maintains a table that records all runs.
- Backs up the Apache Web Server *error.log* file.

Performance Tuning

The following sections describe the features that typically pose challenges or provide significant opportunity for tuning and improvement. Details on the changes with the most effect on throughput and response time are highlighted.

Opcode Caching

PHP Web applications perform many tasks repeatedly — often with the same result each time. Opcode caching allows the runtime to remember the results of a previous invocation of a task and use those results the next time a request is made. The results are stored in a least recently used (LRU) cache that can be sized in the *php.ini* configuration file. During the testing effort, the PHP opcode and variable cache, XCache, was used rather than the Alternative PHP Cache (APC). While XCache is not yet available in binary form, it is easy to build. After the build completes, the *xcache.so* file must be installed in the PHP 5 extensions directory, and the following entries must be added to the PHP configuration file.

```
[xcache-common]
extension = xcache.so

[xcache.admin]
xcache.admin.enable_auth = On
xcache.admin.user = "admin"
xcache.admin.pass = "5f4dcc3b5aa765d61d8327deb882cf99"

[xcache]
xcache.shm_scheme =        "mmap"
xcache.size  =               128M
xcache.count =                40
xcache.slots =               8K
xcache.ttl   =                0
xcache.gc_interval =          0

; same as above but for variable cache
xcache.var_size  =           128M
xcache.var_count =            40
xcache.var_slots =           8K
xcache.var_ttl   =            0
xcache.var_maxttl   =         0
xcache.var_gc_interval =    300
xcache.test =               Off
xcache.mmap_path =    "/dev/zero"
xcache.readonly_protection = Off
```

It is recommended that one cache be used for each CPU in use. During testing efforts, `xcache.count` and `xcache.var_count` were set to the number of CPUs, and the opcode and variable cache size, `xcache.size`, was set to 128 MB. During the testing effort, plenty of memory available and using the recommended memory configuration was not a problem. Using XCache lessens the burden on processors, helping to reduce latency and increase throughput. In order to use XCache, the Lighttpd `max-procs` directive must be set to one. In addition, the number of PHP processes must be configured using the `PHP_FCGI_CHILDREN` environment variable, which is set in the Lighttpd configuration file.

Table 2. Throughput for different numbers of users when XCache is enabled and disabled

Number of Concurrent Users	XCache Enabled	Operations/Second
1,500	No	280
1,750	No	318
2,000	No	320
1,500	Yes	290
1,750	Yes	335
2,000	Yes	352

Table 2 and Figure 3 through Figure 6 show that enabling XCache helps increase throughput, and has greater impact as the number of concurrent users rises. Average response times improved in all tests, with some operations benefiting more from opcode caching. CPU idle time increased as well — moving from an average of 25 percent idle to 55 percent idle with XCache enabled, and reducing %USR by 50 percent. These results are a reflection of the reduction in work required in the PHP application due to caching effects.

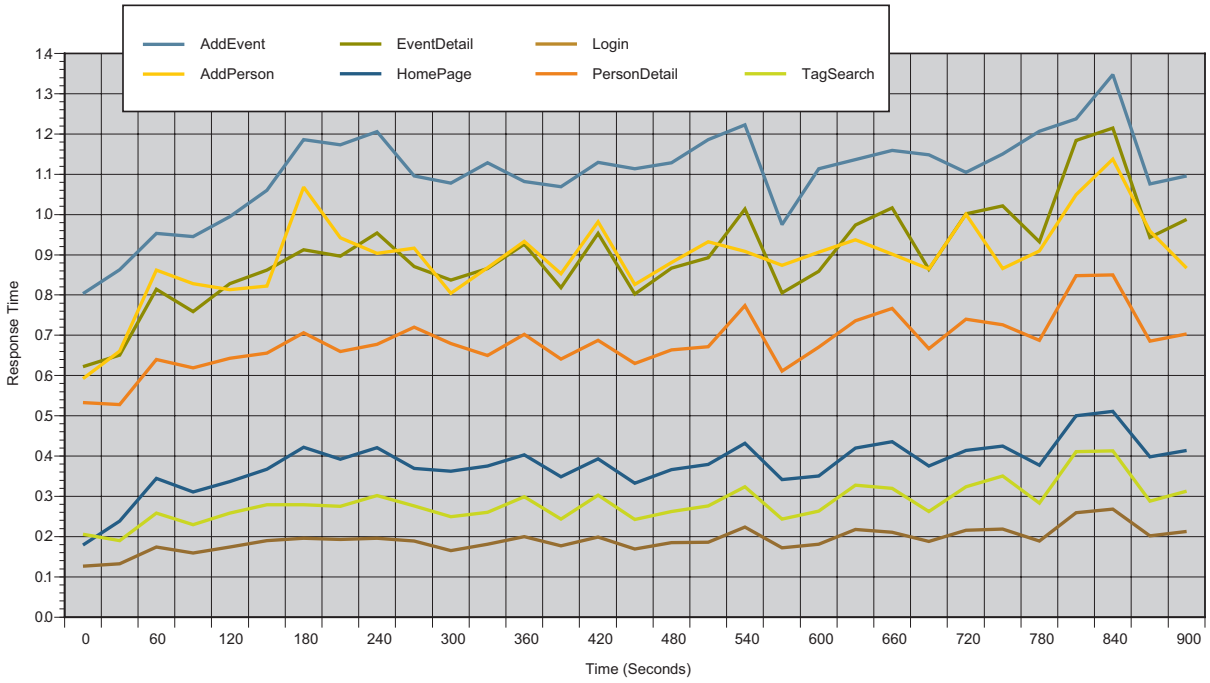

Figure 3. Response time for 1,750 concurrent users with XCache disabled

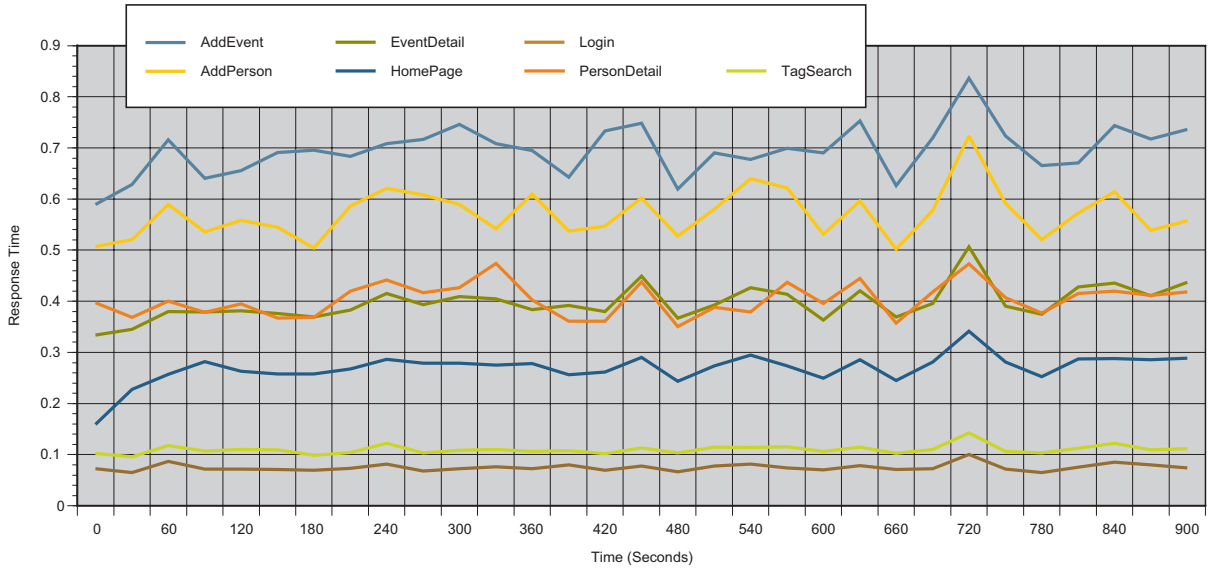

Figure 4. Response time for 1,750 concurrent users with XCache enabled

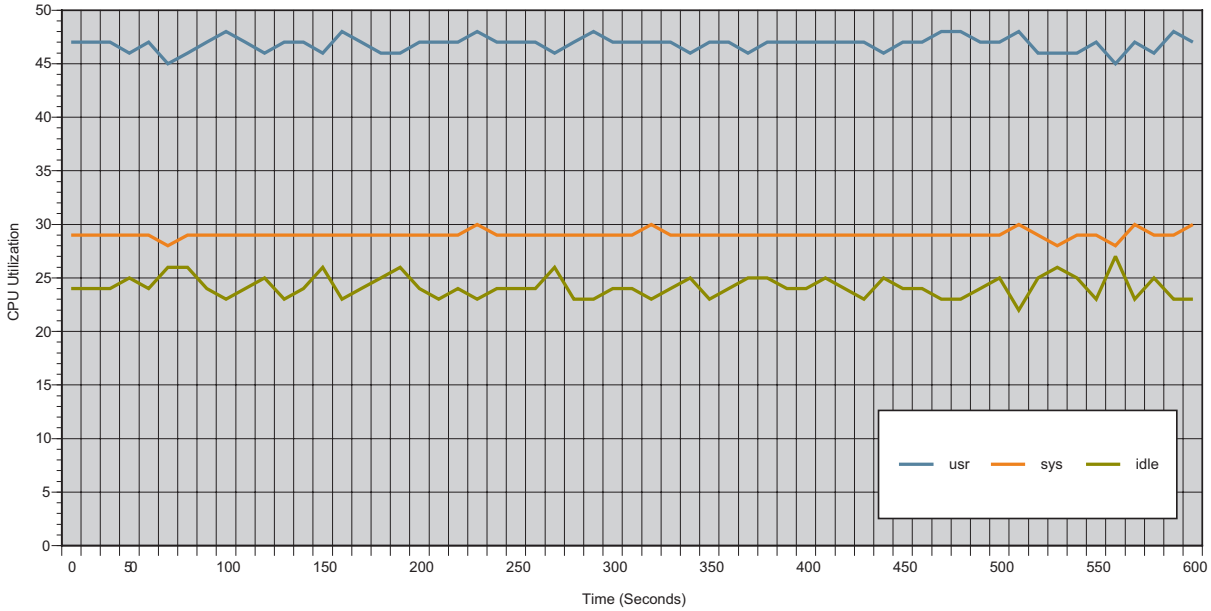

Figure 5. CPU utilization for 1,750 concurrent users with XCache disabled

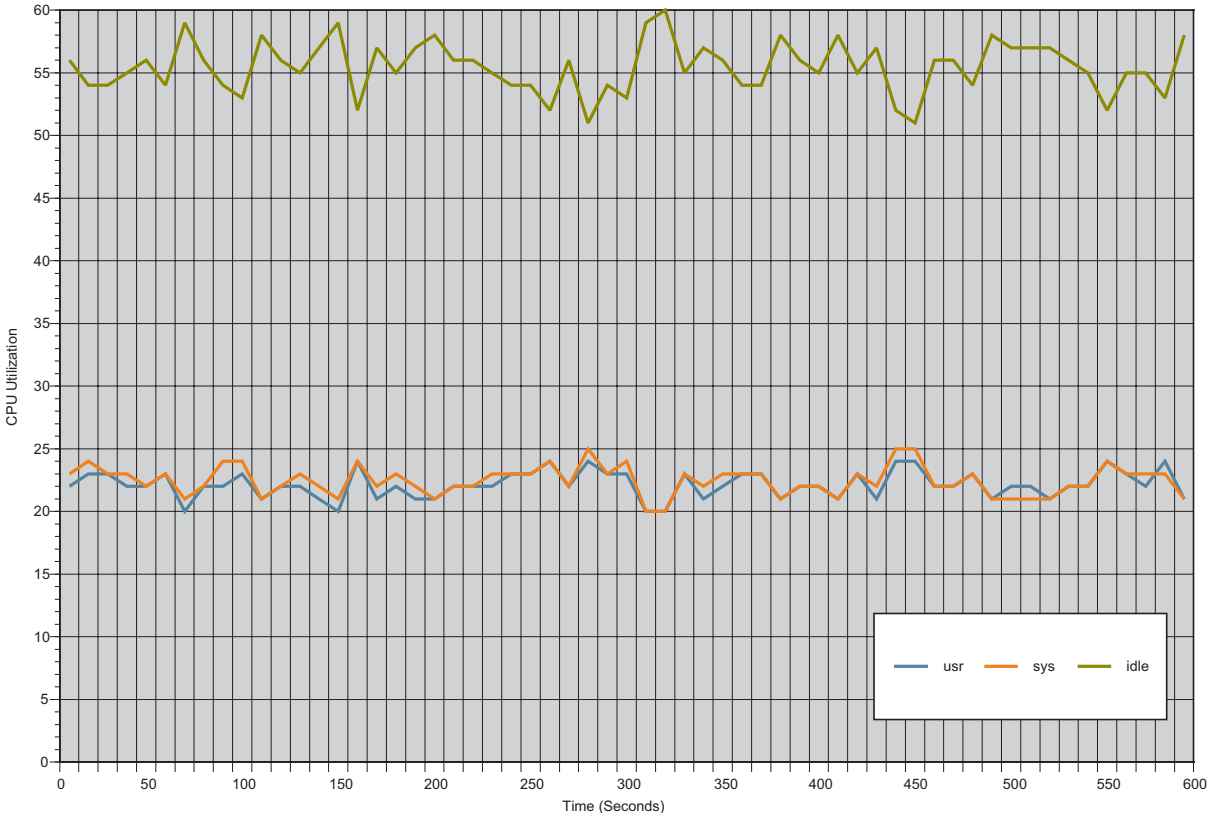

Figure 6. CPU utilization for 1,750 concurrent users with XCache enabled

Temporary Files

Related testing using the Apache Web Server revealed that PHP writes data from file uploads into the */var/tmp* directory. When using loads consistent with this testing effort, the file uploads resulted in considerable amounts of I/O. To alleviate the issue, PHP was configured to write temporary data to the */tmp* directory, which is stored entirely in memory in the Solaris OS. Initial testing showed the Lighttpd Web server also writes file upload data to the */var/tmp* directory, with slow results due to the reading and writing of temporary files. As a result, the Lighttpd Web server was configured to use the */tmp* directory to store temporary files.

```
Lighttpd configuration file options:
  server.upload-dirs = (“/tmp”)

PHP configuration file options:
  session.save_path = /tmp
  upload_tmp_dir = /tmp
  upload_max_fileize = 10M
```

Using the */tmp* directory as a temporary file store for these kinds of workloads can consume a large amount of memory. When memory limits are reached, it is important to add more memory rather than reverting to disk-based temporary file storage in order to maintain the performance benefits.

Sizing the Number of Lighttpd Web Server and PHP Processes

A server with 64 virtual CPUs can run a large number of Lighttpd Web server and PHP back-end processes. Yet finding the right number of processes to run for optimal performance is an iterative process, particularly as the number of concurrent users is increased. While the right number is often found through experimentation, several simple heuristics can help speed the process.

- If single-threaded processes are used, such as occurs with Lighttpd Web server and PHP, make sure the total number of processes is the same or greater than the number of CPUs.
- In general, processes like Web server workers and PHP back-ends do not spend all their time executing on CPUs. As a result, it is possible to run more processes than the number of CPUs in the system. Experimentation can help determine the increased percentage of processes that results in optimum performance.

Table 3 lists a sample of the tests run at the midpoint of the testing effort, prior to the best results being observed. In general, these results show that 12 to 16 Lighttpd Web server processes and 64 to 128 PHP processes yield an optimal configuration.

Table 3. Throughput for different numbers of Lighttpd Web server and PHP processes

Number of Concurrent Users	Number of Lighttpd Processes	Number of PHP Processes	Operations/Second
1,500	16	128	250
1,500	32	128	122
1,500	20	128	240
1,500	12	128	276
1,750	12	128	322
1,750	12	64	320
1,750	12	32	260

The following Lighttpd Web server configuration options were used during the testing effort. The `max-procs` value determines how many parent PHP processes can run, while the `PHP_FCGI_CHILDREN` value determines how many child processes each PHP parent process can spawn. The testing efforts specified one parent and 64 children processes. Running with only one parent process comes with risk. If the parent process fails, all of the PHP processes also fail. However, the configuration as defined lets the Lighttpd Web server restart all of the back-end PHP processes automatically.

```
fastcgi.server =( ".php" =>
                  ( "localhost" =>
                    (
                      "socket" => "/tmp/php-fastcgi.socket",
                      "bin-path" => "/opt/coolstack/php5/bin/php-cgi
                      -c /opt/coolstack/php5/lib/php.ini",
                      "max-procs" => 1,
                      "bin-environment" => (
                        "PHP_FCGI_CHILDREN" => "64",
                        "PHP_FCGI_MAX_REQUESTS" => "10000"
                      ),
                      "broken-scriptfilename" => "enable"
                    )
                  )
                )
server.max-worker = 12
```

The Lighttpd Web server can also be configured to use a socket file to communicate with back-end processes that were started manually using the `spawn-fcgi` command. During the testing effort, the `fastcgi.server` entry in the Lighttpd Web server configuration file was changed to enable the use of socket files as shown below.

```
fastcgi.server =( ".php" =>
                  ( "localhost" =>
                    (
                      "socket" => "/tmp/php-fastcgi.socket",
                      "broken-scriptfilename" => "enable"
                    )
                  )
                )
```

Once the configuration file was modified, the PHP processes were started with the `spawn-fcgi` command.

```
# /opt/coolstack/lighttpd/bin/spawn-fcgi -f "/opt/coolstack/php5/bin
  /php-cgi -c /opt/coolstack/php5/lib/php.ini"
  -s /tmp/php-fastcgi.socket -C 64 -u webservd -g webservd
```

When the number of Lighttpd Web server processes was raised to 18, CPU utilization increased, without benefit to throughput. Figure 7 and Figure 8 show the CPU utilization experienced with 12 and 18 Lighttpd Web server processes. These results indicate the optimum value is in the range of 12 to 16 processes.

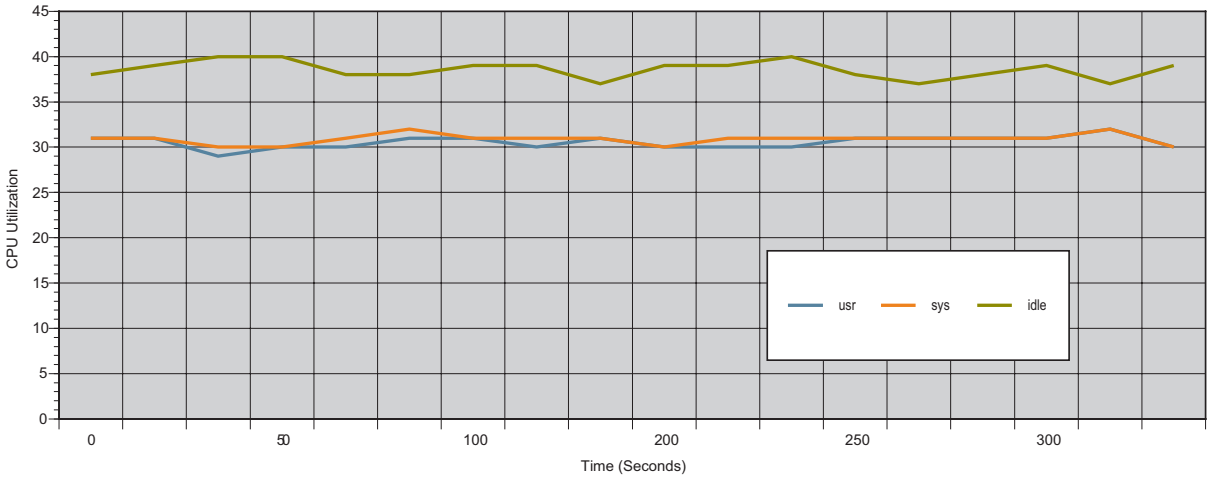

Figure 7. CPU utilization with 12 Lighttpd processes

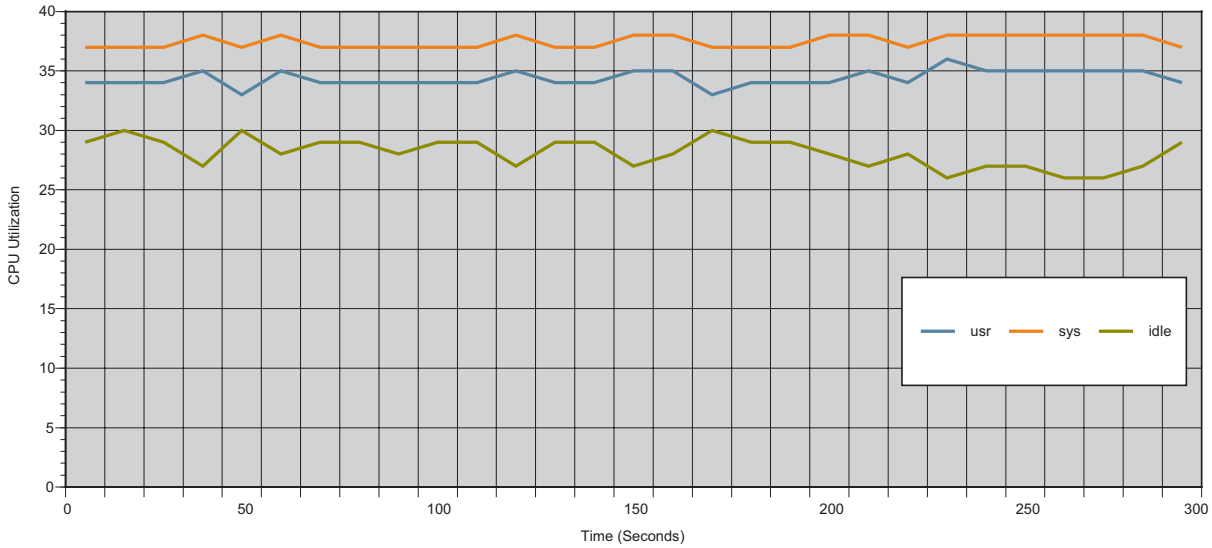

Figure 8. CPU utilization with 18 Lighttpd processes

Database Configuration

The default configuration for the MySQL™ database is not set for large-scale testing. For example, the default value for the maximum number of open database connections, `max_connections`, is 100. The actual value needed depends on the number of PHP processes communicating with the database, and the number of connections each PHP process uses.

For the testing effort, a single instance of the MySQL database was used, and the MyISAM tablestore was used for all runs. The content of the MySQL configuration file, */etc/my.cnf*, used for testing is shown below.

```
[client]
port=3306
socket=/tmp/mysql.sock

[mysqld]
port=3306
socket=/tmp/mysql.sock
key_buffer_size=16M
max_allowed_packet=8M
max_connections=4000
long_query_time=1
query_cache_size=20M
query_cache_limit=2M

[mysqldump]
```

Lighttpd Web Server Configuration

In addition to the number of Lighttpd Web server and PHP processes, a variety of configuration parameters can be controlled by directives in the Lighttpd Web server configuration file. The testing effort utilized Lighttpd Web server version 1.4.18, with a configuration based on the default configuration file that comes with the Cool Stack version of Lighttpd. Any changes to those defaults are detailed within this document. The `server.network-backend` parameter was set to `writev (the default)`, as the Lighttpd Web server tries to use the specified `server.network-backend` for all operations, including sharing file data with back-end processes via UNIX® domain sockets (UDS). Note that the Solaris OS `sendfilev` function was not used since it does not support writing to a UNIX domain socket.

```
server.network-backend = “writev”
```

Memcached

Memcached is a vital part of the Web 2.0 test, and is used by the PHP components of the Web application to cache frequently accessed data. In effect, Memcached is a distributed HashMap designed to act as a LRU cache. Applications must be modified to access Memcached through a client library. Only the number of Memcached instances can be tuned in the version of the Web 2.0 Kit that was used. It can be configured through the application configuration file or the Faban harness. Testing shows that the use of two, three, and four Memcached instances yields the same results for identical runs of this workload at the load levels tested.

Network Interface Card Interrupt Handling

A significant side effect of the large network pipes made available by Gigabit Ethernet and 10 Gigabit Ethernet network interface cards is the additional work that is needed to handle incoming packets. Traditionally every packet of interest to the system generates a hardware interrupt that must be serviced by a CPU. In addition, the CPU must run an interrupt thread that moves the packet up the stack to make it available to the application. Note that each network interface card in Sun SPARC Enterprise T5120 servers includes a CPU that is assigned to handle interrupts.

The `mdb` command can be used to determine which CPU is servicing a network interface card. The following example shows that the interrupts for device `e1000g#1` are serviced by CPU 59. In a Sun SPARC Enterprise T5120 server, CPU 59 is a single thread of one of the processor cores (a virtual CPU).

```
# echo "::interrupts" | mdb -k
Device      Shared  Type      MSG # State   INO     Mondo    Pil    CPU
e1000g#3       no   MSI         4   enbl    0x1c    0x1c       6     61
e1000g#1       no   MSI         3   enbl    0x1b    0x1b       6     59
e1000g#2       no   MSI         2   enbl    0x1a    0x1a       6     58
e1000g#0       no   MSI         1   enbl    0x19    0x19       6     31
```

The `mpstat` command can be used to look at the work being performed by the CPUs. In the example output below, CPUs 58, 59, and 61 are servicing network interface cards. However, all network traffic is coming in on device `e1000g#1`, resulting in high system time (`sys`) and a high number of interrupts (`intr`) and interrupt threads (`ithr`).

```
CPU minf mjf  xcal   intr   ithr    csw   icsw   migr  smtx  srw syscl   usr  sys  wt   idl
58  162  0    4610   369    0       855   31     116   2613  1   13778   32   30   0    39
59  2    0    12626  13438  13321   2600  216    149   2497  0   772     2    70   0    28
60  11   0    665    2260   1       4784  428    385   1796  1   3968    11   26   0    63
```

Note that it is not possible to spread the handling of interrupts across multiple virtual CPUs when the built-in `e1000g` network interface card is used. However, it is possible to distribute the execution of threads that move packets up the stack to applications.

The Solaris OS now includes a device driver architecture for NICs called the Generic LAN Driver (`GLD`). GLD version 3 (`GLDv3`) includes features that help improve the performance of NICs in the Solaris 10 OS and OpenSolaris operating system. In addition, `GLDv3` adds support for soft rings—virtual receive queues for incoming network packets. Soft rings have worker threads that process incoming packets, and each thread has an affinity to a specific CPU. When an incoming packet generates an interrupt, the thread handling the interrupt passes packet handling to the worker thread. This spreads the overhead of moving packets up the stack to the application across multiple CPUs. The overhead of the initial interrupt handling done by a single CPU remains. However, the burden on that CPU is reduced.

The number of soft rings can be configured by adding the following line to the */etc/system* file and rebooting the system.

```
set ip:ip_soft_ring_cnt=8
```

Sun systems with CMT designs are configured from the factory with a default `soft_rings_cnt` of 16. This is set in `/etc/system` as described above. Changes to */etc/system* generally survive system upgrades unless those upgrades are made using automated mechanisms such as the Solaris Jumpstart™ software. Therefore it is important to check the value for `soft_rings_cnt` on the system as it may have been inadvertently reset to the Solaris OS default of two.

The `ndd -set` command can be used to set the number of soft rings. After the `ndd` command completes, it is important to unplumb and plumb the interface in order for the changes to take effect. Note this change is lost when the system is rebooted. Modify the */etc/system* file to make the change permanent.

```
ndd -set /dev/ip ip_soft_rings_cnt 8
```

The current number of soft rings can be determined using the `ndd -get` command.

```
ndd -get /dev/ip ip_soft_rings_cnt
```

Testing efforts showed the optimum soft rings setting for the test workload — and perhaps for any Web application handling a large number of concurrent users on Sun SPARC Enterprise T5120 servers — is eight to sixteen. Table 4 shows the results from test runs using two and eight soft ring configurations. These results indicate that the benefit of increasing the soft ring count rises with higher workloads. Tests were run with sixteen soft rings, but no additional benefits were observed over running with eight soft rings.

Table 4. Throughput comparison for two and eight soft rings

Number of Concurrent Users	ip_soft_ring_cnt	Operations/Second
1,000	2	195
1,500	2	290
2,000	2	344
2,250	2	330
1,000	8	198
1,500	8	297
2,000	8	385
2,250	8	416

All Sun servers with chip multithreading technology have soft rings enabled by default, with two soft rings defined. Soft rings can be disabled by adding the following line to the */etc/system* file and rebooting the system.

```
set ip_squeue_soft_ring=0
```

Setting the number of soft rings to zero reduced throughput to 167 operations/second with 2,000 concurrent users during the testing effort. Table 5 identifies the soft rings settings, overall CPU utilization, and specific statistics for the CPU servicing the interrupts for the NIC doing the work. These results suggest that increasing the number of soft rings increases the number of voluntary and involuntary context switches experienced on the system.

Table 5. Settings, utilization values, and statistics

ip_squeue_soft_ring	1	1	1	1
lp:ip_soft_rings_cnt	2	8	16	64
%SYS	28	30	31	31
%USR	26	27	27	27
%IDLE	46	43	42	42
Context Switches	119859	145142	152788	152714
Involuntary Context Switches	5857	7429	10047	14408
Interrupts	79296	91046	95427	95194
Interrupt Threads	16147	1627	16062	15770
System Calls	692904	691896	666325	686565
CPU Cross Calls	336424	353031	363367	355585
CPU Migrations	24371	28642	31986	162135

Nagle's Algorithm

Applications that generate large numbers of very small packets tend to suffer from significant overhead. Nagle's Algorithm helps reduce network overhead by batching packets and sending them together. For example, TCP/IP packets can carry a variable amount of data. However, each TCP/IP packet contains the same sized header (40 bytes). As a result, sending 1,000 packets that are each 1 byte in size is 40 times less efficient than sending a single packet with 1,000 bytes.

With Nagle's Algorithm enabled, the first small packet is sent. However, subsequent packets are not sent until an acknowledgment is received for the first packet, or until there is a full packet of data to send. While such a mechanism can work well, the delayed acknowledgment feature of the Transmission Control Protocol (TCP) can impact results. This feature acknowledges the receipt of a packet when it has data to return to the caller, when it receives a second packet from the caller, or a specified time has elapsed. In this scenario, Nagle's Algorithm does not get the immediate acknowledgment it expects, adding delay to the round trip time of an HTTP request/response pair.

Nagle's Algorithm used to be enabled by default on most operating systems. While it is enabled in the Solaris OS, it is disabled in the Linux environment. Furthermore, most Web servers disable Nagle's Algorithm at the socket layer by creating new sockets with the `TCP_NODELAY` option. However, the Lighttpd Web server does not disable Nagle's Algorithm by default. As a result, running the Lighttpd Web server on the Solaris OS creates an environment with Nagle's Algorithm enabled.

The Lighttpd Web server packages included in the Cool Stack software disable Nagle's Algorithm by using the `TCP_NODELAY` option when creating sockets. During the testing effort, versions of the Lighttpd Web Server that do and do not disable Nagle's Algorithm were compared. Based on the results shown in Table 6, the recommendation is to disable Nagle's Algorithm for Web-based workloads.

Table 6. Comparison of 90 percent response times with and without Nagle's Algorithm

Number of Concurrent Users	Nagle Enabled	90 Percent RT for HomePage
1,500	No	1.20
1,750	No	1.20
2,000	No	1.35
1,500	Yes	0.30
1,750	Yes	0.45
2,000	Yes	0.60

Network I/O

Network I/O is often a limiting factor in Web server environments. Gigabit Ethernet supports full duplex operation, and can handle up to 1 Gigabit per second inbound and outbound. These theoretical limits are reduced in practice by the overhead associated in transmitting data across the network, and are dependent on the hardware in use. During the testing effort, outbound traffic (HTTP responses) was three times greater than inbound traffic (HTTP requests and file uploads). As a result, link saturation was expected on the outbound side of transactions. Indeed, testing results showed approximately 1 Gigabit per second was reached with 2,250 users as evidenced by the following `nicstat` command output. These results reveal outbound throughput of 117,000 KB/second (914 megabits/second) outbound, and 45,000 KB/second (351 megabits/second) inbound.

```
Time      Int       rKB/s     wKB/s     rPk/s     wPk/s     rAvs    wAvs    %Util   Sat
22:54:07  lo0       0.00      0.00      4377.0    4377.0    0.00    0.00    0.00    0.00
22:54:07  e1000g0   0.15      0.01      2.40      0.20      64.00   64.00   0.00    0.00
22:54:07  e1000g1   45305.4   116290    50417.5   93264.8   920.2   1276.8  100     0.00

Time      Int       rKB/s     wKB/s     rPk/s     wPk/s     rAvs    wAvs    %Util   Sat
22:54:17  lo0       0.00      0.00      4525.2    4525.2    0.00    0.00    0.00    0.00
22:54:17  e1000g0   0.19      0.00      2.60      0.00      74.85   0.00    0.00    0.00
22:54:17  e1000g1   42564.8   117343    48242.1   93275.1   903.5   1288.2  100     0.00

Time      Int       rKB/s     wKB/s     rPk/s     wPk/s     rAvs    wAvs    %Util   Sat
22:54:27  lo0       0.00      0.00      4379.0    4379.0    0.00    0.00    0.00    0.00
22:54:27  e1000g0   0.15      0.00      1.90      0.00      78.84   0.00    0.00    0.00
22:54:27  e1000g1   42850.0   116126    48580.6   92562.4   903.2   1284.7  100     0.00
```

Figure 9 shows the relationship of outbound network utilization versus the number of concurrent users.

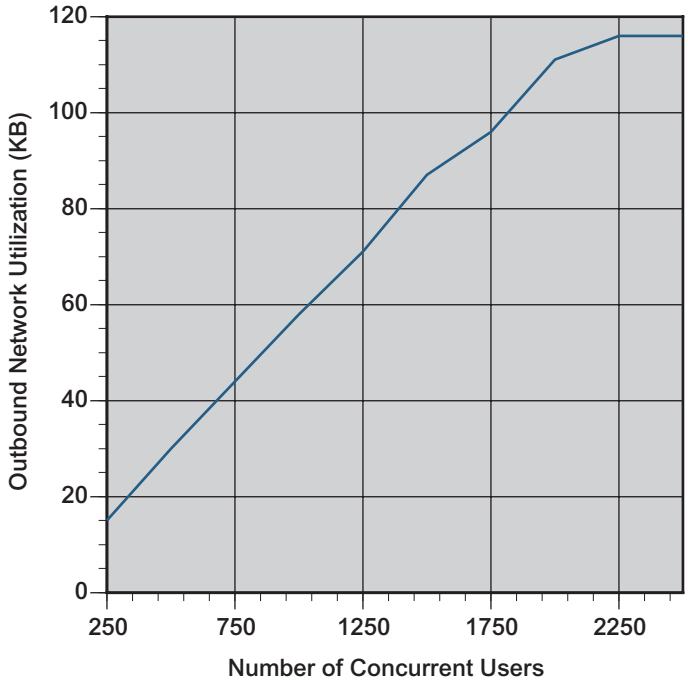

Figure 9. Outbound network utilization versus the number of concurrent users

Best Practices for Deployment

The testing effort demonstrates that 2,250 user connections running a representative workload can be maintained while achieving approximately 92 percent of the maximum throughput of 450 operations/second. At the same time, the average and 90th percentile response times for all operations were within fairly aggressive targets as defined by the Web 2.0 Kit. The network interface became saturated with 2,250 users. Testing stopped at this level. It is possible to use the multiple networking interfaces with link aggregation in the Sun SPARC Enterprise T5120 server and scale further.

Figure 10 shows the progression of the testing effort, and highlights where key configuration changes were made that led to the next level of performance.

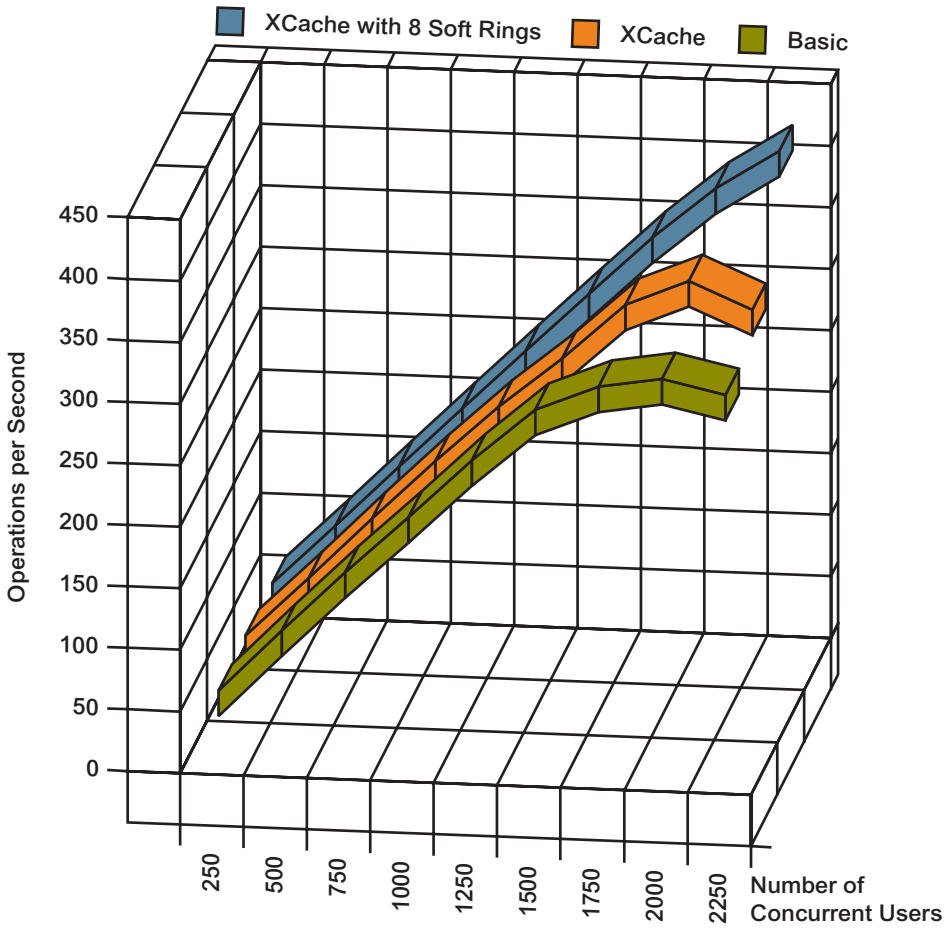

Figure 10. Operations per second versus the number of concurrent users

Several best practices can be implemented to improve the performance of the Lighttpd Web server when running on Sun SPARC Enterprise T5120 servers.

- *Use /tmp for temporary file storage*
  The Lighttpd Web server and PHP store file uploads to the */var/tmp* directory by default, resulting in significant file I/O that bottlenecks the I/O subsystem. Using the */tmp* directory for temporary storage reduces pressure on the I/O subsystem.
- *Increase available memory when using /tmp for temporary file storage*
  Using the */tmp* directory for temporary file storage requires additional memory. Many Web workloads cannot scale without moving temporary storage to the */tmp* directory. Be sure to provide sufficient memory in the system to experience optimum performance.
- *Use an opcode cache*
  Using an opcode in an environment with high workloads can help improve throughput and decrease response time for many operations. During the testing effort, use of an opcode cache improved throughput by 10 percent.

- *Increase the number of soft rings*
  The factory setting for soft rings is 16, but upgrading or reinstalling the Solaris Operating System can result in this setting being reset to two, a setting that is inadequate for network-intensive applications. Be sure to check and increase the number of soft rings for high workload conditions, if necessary.
- *Disable Nagle's Algorithm*
  Using Nagle's Algorithm with certain types of workloads can delay the sending of packets and acknowledgments. Disable Nagle's Algorithm for Web-based workloads to help improve network throughput and response time.
- *Optimize the number of Lighttpd Web server processes*
  Configuring the system to use between 12 and 16 Lighttpd Web server processes can help optimize performance on Sun SPARC Enterprise T5120 servers.

Conclusion

Sun SPARC Enterprise T5120 servers provide a solid platform for Web workloads, particularly those that utilize the Lighttpd Web server. Blending the performance and scalability of midrange servers with the economies of energy-efficient CMT designs, these servers support up to 128 simultaneous execution threads on a single processor, large memory, and integrated on-chip I/O technology to deliver the compute power and networking capacity demanded by Web 2.0 architectures.

About the Author

Amanda Waite is a Staff Engineer in Sun's ISV engineering team. Since joining Sun in 1998, Amanda has helped ISVs and systems integrators optimize products and solutions for Sun platforms. She now works with the open source communities, including the Lighttpd community, in a similar fashion. Amanda's core expertise is in the Java environment and scripting languages, such as Ruby, the JavaScript programming language, and PHP. Currently, Amanda is working on integrating the Lighttpd Web server into the OpenSolaris project.

References

Faban Harness:
`http://faban.sourcenet.net`

Sun SPARC Enterprise T5120 Servers:
`http://sun.com/servers/coolthreads/t5120`

Lighttpd Web Server:
`http://www.lighttpd.net`

Cool Stack:
`http://cooltools.sunsource.net/coolstack`

Add Web 2.0 Kit:
`http://cooltools.sunsource.net/web20kit`

Ordering Sun Documents

The SunDocs℠ program provides more than 250 manuals from Sun Microsystems, Inc. If you live in the United States, Canada, Europe, or Japan, you can purchase documentation sets or individual manuals through this program.

Accessing Sun Documentation Online

The `docs.sun.com` web site enables you to access Sun technical documentation online. You can browse the `docs.sun.com` archive or search for a specific book title or subject. The URL is `http://docs.sun.com/`

To reference Sun BluePrints Online articles, visit the Sun BluePrints Online Web site at:
`http://www.sun.com/blueprints/online.html`

Sun Microsystems, Inc. 4150 Network Circle, Santa Clara, CA 95054 USA **Phone** 1-650-960-1300 or 1-800-555-9SUN (9786) **Web** sun.com

© 2008 Sun Microsystems, Inc. All rights reserved. Sun, Sun Microsystems, the Sun logo, CoolThreads, Java, JavaScript, JumpStart, MySQL, OpenSolaris, Solaris, SunDocs, Sun BluePrints, and Sun Fire are trademarks or registered trademarks of Sun Microsystems, Inc. or its subsidiaries in the United States and other countries. All SPARC trademarks are used under license and are trademarks or registered trademarks of SPARC International, Inc. in the US and other countries. Products bearing SPARC trademarks are based upon an architecture developed by Sun Microsystems, Inc.UNIX is a registered trademark in the United States and other countries, exclusively licensed through X/Open Company, Ltd. Information subject to change without notice. Printed in USA 09/08