PerformanceScalingUp

Performance Optimization: Scaling Up

This wiki page is intended to wikify: http://people.apache.org/~sctemme/ApconUS2007/scalingup.pdf

Drop all of the things that are out-dated, or, if applicable update them.

Should we have managed to reach some kind of consensus on docs@, we can then merge it into

http://httpd.apache.org/docs/trunk/misc/perf-tuning.html

See also: http://wiki.apache.org/httpd/PerformanceScalingOut

Acknowledgment

This documentation is based on the ApacheCon Presentations on Performance and Scalability, by Sander S. Temme. The original PDFs can be found at: http://people.apache.org/~sctemme/ApconUS2007/

Introduction

The Performance Tuning page in the Apache 1.3 documentation says:

“Apache is a general webserver, which is designed to be correct first, and fast second. Even so, its performance is quite satisfactory. Most sites have less than 10Mbits of outgoing bandwidth, which Apache can fill using only a low end Pentium-based webserver.”

However, this sentence was written a few years ago, and in the meantime several things have happened. On one hand, web server hardware has become much faster. On the other hand, many sites now are allowed much more than ten megabits per second of outgoing bandwidth. In addition, web applications have become more complex. The classic brochureware site is alive and well, but the web has grown up substantially as a computing application platform and webmasters may find themselves running dynamic content in Perl, PHP or Java, all of which take a toll on performance.

Therefore, in spite of strides forward in machine speed and bandwidth allowances, web server performance and web application performance remain areas of concern. In this documentation several aspects of web server performance will be discussed.

What Will and Will Not Be Discussed

The session will focus on easily accessible configuration and tuning options for Apache httpd 2.2 and 2.3 as well as monitoring tools. Monitoring tools will allow you to observe your web server to gather information about its performance, or lack thereof. We'll assume that you don't have an unlimited budget for server hardware, so the existing infrastructure will have to do the job. You have no desire to compile your own Apache, or to recompile the operating system kernel. We do assume, though, that you have some familiarity with the Apache httpd configuration file.

Monitoring Your Server

The first task when sizing or performance-tuning your server is to find out how your system is currently performing. By monitoring your server under real-world load, or artificially generated load, you can extrapolate its behavior under stress, such as when your site is mentioned on Slashdot.

Monitoring Tools

top

The top tool ships with Linux and FreeBSD. Solaris offers `prstat'. It collects a number of statistics for the system and for each running process, then displays them interactively on your terminal. The data displayed is refreshed every second and varies by platform, but typically includes system load average, number of processes and their current states, the percent CPU(s) time spent executing user and system code, and the state of the virtual memory system. The data displayed for each process is typically configurable and includes its process name and ID, priority and nice values, memory footprint, and percentage CPU usage. The following example shows multiple httpd processes (with MPM worker and event) running on an Linux (Xen) system:

```
top - 23:10:58 up 71 days,  6:14,  4 users,  load average: 0.25, 0.53, 0.47
Tasks: 163 total,   1 running, 162 sleeping,   0 stopped,   0 zombie
Cpu(s): 11.6%us,  0.7%sy,  0.0%ni, 87.3%id,  0.4%wa,  0.0%hi,  0.0%si,  0.0%st
Mem:   2621656k total,  2178684k used,   442972k free,   100500k buffers
Swap:  4194296k total,   860584k used,  3333712k free,  1157552k cached

  PID USER      PR  NI  VIRT  RES  SHR S %CPU %MEM    TIME+  COMMAND
16687 example_  20   0 1200m 547m 179m S   45 21.4   1:09.59 httpd-worker
15195 www       20   0  441m  33m 2468 S    0  1.3   0:41.41 httpd-worker
    1 root      20   0 10312  328  308 S    0  0.0   0:33.17 init
    2 root      15  -5     0    0    0 S    0  0.0   0:00.00 kthreadd
    3 root      RT  -5     0    0    0 S    0  0.0   0:00.14 migration/0
    4 root      15  -5     0    0    0 S    0  0.0   0:04.58 ksoftirqd/0
    5 root      RT  -5     0    0    0 S    0  0.0   4:45.89 watchdog/0
    6 root      15  -5     0    0    0 S    0  0.0   1:42.52 events/0
    7 root      15  -5     0    0    0 S    0  0.0   0:00.00 khelper
   19 root      15  -5     0    0    0 S    0  0.0   0:00.00 xenwatch
   20 root      15  -5     0    0    0 S    0  0.0   0:00.00 xenbus
   28 root      RT  -5     0    0    0 S    0  0.0   0:00.14 migration/1
   29 root      15  -5     0    0    0 S    0  0.0   0:00.20 ksoftirqd/1
   30 root      RT  -5     0    0    0 S    0  0.0   0:05.96 watchdog/1
   31 root      15  -5     0    0    0 S    0  0.0   1:18.35 events/1
   32 root      RT  -5     0    0    0 S    0  0.0   0:00.08 migration/2
   33 root      15  -5     0    0    0 S    0  0.0   0:00.18 ksoftirqd/2
   34 root      RT  -5     0    0    0 S    0  0.0   0:06.00 watchdog/2
   35 root      15  -5     0    0    0 S    0  0.0   1:08.39 events/2
   36 root      RT  -5     0    0    0 S    0  0.0   0:00.10 migration/3
   37 root      15  -5     0    0    0 S    0  0.0   0:00.16 ksoftirqd/3
   38 root      RT  -5     0    0    0 S    0  0.0   0:06.08 watchdog/3
   39 root      15  -5     0    0    0 S    0  0.0   1:22.81 events/3
   68 root      15  -5     0    0    0 S    0  0.0   0:06.28 kblockd/0
   69 root      15  -5     0    0    0 S    0  0.0   0:00.04 kblockd/1
   70 root      15  -5     0    0    0 S    0  0.0   0:00.04 kblockd/2
```

Top is a wonderful tool even though it's slightly resource intensive (when running, its own process is usually in the top ten CPU gluttons). It is indispensable in determining the size of a running process, which comes in handy when determining how many server processes you can run on your machine. How to do this is described in 'sizing MaxClients'. Top is, however, an interactive tool and running it continuously has few if any advantages.

free

This command is only available on Linux. It shows how much memory and swap space is in use. Linux allocates unused memory as file system cache. The free command shows usage both with and without this cache. The free command can be used to find out how much memory the operating system is using, as described in the paragraph 'Sizing MaxClients'. The output of free looks like this:

```
sctemme@brutus:~$ free
             total       used       free     shared    buffers     cached
Mem:       4026028    3901892     124136          0     253144     841044
-/+ buffers/cache:    2807704    1218324
Swap:      3903784      12540    3891244
```

vmstat

This command is available on many unix platforms. It displays a large number of operating system metrics. Run without argument, it displays a status line for that moment. When a numeric argument is added, the status is redisplayed at designated intervals. For example, `vmstat 5` causes the information to reappear every five seconds. Vmstat displays the amount of virtual memory in use, how much memory is swapped in and out each second, the number of processes currently running and sleeping, the number of interrupts and context switches per second and the usage percentages of the CPU.

The following is `vmstat` output of an idle server:

```
[sctemme@GayDeceiver sctemme]$ vmstat 5 3
   procs                      memory     swap         io    system        cpu
 r b w     swpd   free   buff cache si so       bi    bo in     cs us  sy id
 0 0 0        0 186252   6688 37516    0    0   12     5 47    311  0   1 99
 0 0 0        0 186244   6696 37516    0    0    0    16 41    314  0   0 100
 0 0 0        0 186236   6704 37516    0    0    0     9 44    314  0   0 100
```

And this is output of a server that is under a load of one hundred simultaneous connections fetching static content:

```
[sctemme@GayDeceiver sctemme]$ vmstat 5 3
   procs                      memory     swap    io      system       cpu
 r b w     swpd   free   buff cache si so     bi bo   in     cs us sy  id
 1 0 1        0 162580   6848 40056    0    0 11  5 150     324  1  1  98
 6 0 1        0 163280   6856 40248    0    0  0 66 6384 1117   42 25  32
11 0 0        0 162780   6864 40436    0    0  0 61 6309 1165   33 28  40
```

The first line gives averages since the last reboot. The subsequent lines give information for five second intervals. The second argument tells vmstat to generate three reports and then exit.

SE Toolkit

The SE Toolkit is a system monitoring toolkit for Solaris. Its programming language is based on the C preprocessor and comes with a number of sample scripts. It can use both the command line and the GUI to display information. It can also be programmed to apply rules to the system data. The example script shown in Figure 2, Zoom.se, shows green, orange or red indicators when utilization of various parts of the system rises above certain thresholds. Another included script, Virtual Adrian, applies performance tuning metrics according to.

The SE Toolkit has drifted around for a while and has had several owners since its inception. It seems that it has now found a final home at Sunfreeware.com, where it can be downloaded at no charge. There is a single package for Solaris 8, 9 and 10 on SPARC and x86, and includes source code. SE Toolkit author Richard Pettit has started a new company, Captive Metrics4 that plans to bring to market a multiplatform monitoring tool built on the same principles as SE Toolkit, written in Java.

DTrace

Given that DTrace is available for Solaris, FreeBSD and OS X, it might be worth exploring it. There's also mod_dtrace available for httpd.

mod_status

The mod_status module gives an overview of the server performance at a given moment. It generates an HTML page with, among others, the number of Apache processes running and how many bytes each has served, and the CPU load caused by httpd and the rest of the system. The Apache Software Foundation uses mod_status on its own web site. If you put the `ExtendedStatus On` directive in your `httpd.conf`, the `mod_status` page will give you more information at the cost of a little extra work per request.

Web Server Log Files

Monitoring and analyzing the log files httpd writes is one of the most effective ways to keep track of your server health and performance. Monitoring the error log allows you to detect error conditions, discover attacks and find performance issues. Analyzing the access logs tells you how busy your server is, which resources are the most popular and where your users come from. Historical log file data can give you invaluable insight into trends in access to your server, which allows you to predict when your performance needs will overtake your server capacity.

Error Log

The error log will contain messages if the server has reached the maximum number of active processes or the maximum number of concurrently open files. The error log also reflects when processes are being spawned at a higher-than-usual rate in response to a sudden increase in load. When the server starts, the stderr file descriptor is redirected to the error logfile, so any error encountered by httpd after it opens its logfiles will appear in this log. This makes it good practice to review the error log frequently.

Before Apache httpd opens its logfiles, any errors will be written to the stderr stream. If you start httpd manually, this error information will appear on your terminal and you can use it directly to troubleshoot your server. If your httpd is started by a startup script, the destination of early error messages depends on their design. The `/var/log/messages` file is usually a good bet. On Windows, early error messages are written to the Applications Event Log, which can be viewed through the Event Viewer in Administrative Tools.

The Error Log is configured through the `ErrorLog` and `LogLevel` configuration directives. The error log of httpd's main server configuration receives the log messages that pertain to the entire server: startup, shutdown, crashes, excessive process spawns, etc. The `ErrorLog` directive can also be used in virtual host containers. The error log of a virtual host receives only log messages specific to that virtual host, such as authentication failures and 'File not Found' errors.

On a server that is visible to the Internet, expect to see a lot of exploit attempt and worm attacks in the error log. A lot of these will be targeted at other server platforms instead of Apache, but the current state of affairs is that attack scripts just throw everything they have at any open port, regardless of which server is actually running or what applications might be installed. You could block these attempts using a firewall or mod_security, but this falls outside the scope of this discussion.

The `LogLevel` directive determines the level of detail included in the logs. There are eight log levels as described here:

Level	Description
emerg	Emergencies - system is unusable.
alert	Action must be taken immediately.
crit	Critical Conditions.
error	Error conditions.
warn	Warning conditions.
notice	Normal but significant condition.
info	Informational.
debug	Debug-level messages

The default log level is warn. A production server should not be run on debug, but increasing the level of detail in the error log can be useful during troubleshooting. Starting with 2.3.8 `LogLevel` can be specified on a per module basis:

```
LogLevel debug mod_ssl:warn
```

This puts all of the server in debug mode, except for `mod_ssl`, which tends to be very noisy.

Access Log

Apache httpd keeps track of every request it services in its access log file. In addition to the time and nature of a request, httpd can log the client IP address, date and time of the request, the result and a host of other information. The various logging format features are documented in the manual. This file exists by default for the main server and can be configured per virtual host by using the `TransferLog` or `CustomLog` configuration directive.

The access logs can be analyzed with any of several free and commercially available programs. Popular free analysis packages include Analog and Webalizer. Log analysis should be done offline so the web server machine is not burdened by processing the log files. Most log analysis packages understand the Common Log Format. The fields in the log lines are explained in in the following:

```
195.54.228.42 - - [24/Mar/2007:23:05:11 -0400] "GET /sander/feed/ HTTP/1.1" 200 9747
64.34.165.214 - - [24/Mar/2007:23:10:11 -0400] "GET /sander/feed/atom HTTP/1.1" 200 9068
60.28.164.72 - - [24/Mar/2007:23:11:41 -0400] "GET / HTTP/1.0" 200 618
85.140.155.56 - - [24/Mar/2007:23:14:12 -0400] "GET /sander/2006/09/27/44/ HTTP/1.1" 200 14172
85.140.155.56 - - [24/Mar/2007:23:14:15 -0400] "GET /sander/2006/09/21/gore-tax-pollution/ HTTP/1.1" 200 15147
74.6.72.187 - - [24/Mar/2007:23:18:11 -0400] "GET /sander/2006/09/27/44/ HTTP/1.0" 200 14172
74.6.72.229 - - [24/Mar/2007:23:24:22 -0400] "GET /sander/2006/11/21/os-java/ HTTP/1.0" 200 13457
```

Field	Content	Explanation
Client IP	195.54.228.42	IP address where the request originated
RFC 1413 ident		Remote user identity as reported by their identd
username		Remote username as authenticated by Apache

<ac:structured-macro ac:name="unmigrated-wiki-markup" ac:schema-version="1" ac:macro-id="d29ccf4c-329d-40e3-a957-2ddf5a9ddbe5"><ac:plain-text-body><![CDATA[	timestamp	[24/Mar/2007:23:05:11 -0400]	Date and time of request	]]></ac:plain-text-body></ac:structured-macro>
Request	"GET /sander /feed/ HTTP/1. 1"	Request line		
Status Code	200	Response code		
Content Bytes	9747	Bytes transferred w/o headers		

Rotating Log Files

There are several reasons to rotate logfiles. Even though almost no operating systems out there have a hard file size limit of two Gigabytes anymore, log files simply become too large to handle over time. Additionally, any periodic log file analysis should not be performed on files to which the server is actively writing. Periodic logfile rotation helps keep the analysis job manageable, and allows you to keep a closer eye on usage trends.

On unix systems, you can simply rotate logfiles by giving the old file a new name using mv. The server will keep writing to the open file even though it has a new name. When you send a graceful restart signal to the server, it will open a new logfile with the configured name. For example, you could run a script from cron like this:

```
#!/bin/sh
APACHE=/usr/local/apache2
HTTPD=$APACHE/bin/httpd
mv $APACHE/logs/access_log $APACHE/logarchive/access_log-‘date +%F‘
$HTTPD -k graceful
```

This approach also works on Windows, just not as smoothly. While the httpd process on your Windows server will keep writing to the log file after it has been renamed, the Windows Service that runs Apache can not do a graceful restart. Restarting a Service on Windows means stopping it and then starting it again. The advantage of a graceful restart is that the httpd child processes get to complete responding to their current requests before they exit. Meanwhile, the httpd server becomes immediately available again to serve new requests. The stop-start that the Windows Service has to perform will interrupt any requests currently in progress, and the server is unavailable until it is started again. Plan for this when you decide the timing of your restarts.

A second approach is to use piped logs. From the `CustomLog`, `TransferLog`
or `ErrorLog` directives you can send the log data into any program using a pipe character (`|`). For instance:

```
CustomLog "|/usr/local/apache2/bin/rotatelogs /var/log/access_log 86400" common
```

The program on the other end of the pipe will receive the Apache log data on its stdin stream, and can do with this data whatever it wants. The rotatelogs program that comes with Apache seamlessly turns over the log file based on time elapsed or the amount of data written, and leaves the old log files with a timestamp suffix to its name. This method for rotating logfiles works well on unix platforms, but is currently broken on Windows.

Logging and Performance

Writing entries to the Apache log files obviously takes some effort, but the information gathered from the logs is so valuable that under normal circumstances logging should not be turned off. For optimal performance, you should put your disk-based site content on a different physical disk than the server log files: the access patterns are very different. Retrieving content from disk is a read operation in a fairly random pattern, and log files are written to disk sequentially.

Do not run a production server with your error `LogLevel` set to debug. This log level causes a vast amount of information to be written to the error log, including, in the case of SSL access, complete dumps of BIO read and write operations. The performance implications are significant: use the default warn level instead.

If your server has more than one virtual host, you may give each virtual host a separate access logfile. This makes it easier to analyze the logfile later. However, if your server has many virtual hosts, all the open logfiles put a resource burden on your system, and it may be preferable to log to a single file. Use the `%v`
format character at the start of your LogFormat and starting 2.3.8 of your `ErrorLogFormat`
to make httpd print the hostname of the virtual host that received the request or the error at the beginning of each log line. A simple Perl script can split out the log file after it rotates: one is included with the Apache source under `support/split-logfile`.

You can use the `BufferedLogs` directive to have Apache collect several log lines in memory before writing them to disk. This might yield better performance, but could affect the order in which the server's log is written.

Generating A Test Load

It is useful to generate a test load to monitor system performance under realistic operating circumstances. Besides commercial packages such as LoadRunner, there are a number of freely available tools to generate a test load against your web server.

- Apache ships with a test program called ab, short for Apache Bench. It can generate a web server load by repeatedly asking for the same file in rapid succession. You can specify a number of concurrent connections and have the program run for either a given amount of time or a specified number of requests.
- Another freely available load generator is http load11 . This program works with a URL file and can be compiled with SSL support.

- The Apache Software Foundation offers a tool named flood12 . Flood is a fairly sophisticated program that is configured through an XML file.
- Finally, JMeter13 , a Jakarta subproject, is an all-Java load-testing tool. While early versions of this application were slow and difficult to use, the current version 2.1.1 seems to be versatile and useful.
- ASF external projects, that have proven to be quite good: grinder, httperf, tsung, FunkLoad

When you load-test your web server, please keep in mind that if that server is in production, the test load may negatively affect the server's response. Also, any data traffic you generate may be charged against your monthly traffic allowance.

Configuring for Performance

Apache Configuration

The Apache 2.2 httpd is by default a pre-forking web server. When the server starts, the parent process spawns a number of child processes that do the actual work of servicing requests. But Apache httpd 2.0 introduced the concept of the Multi-Processing Module (MPM). Developers can write MPMs to suit the process- or threadingarchitecture of their specific operating system. Apache 2 comes with special MPMs for Windows, OS/2, Netware and BeOS. On unix-like platforms, the two most popular MPMs are Prefork and Worker. The Prefork MPM offers the same pre-forking process model that Apache 1.3 uses. The Worker MPM runs a smaller number of child processes, and spawns multiple request handling threads within each child process. In 2.3+ MPMs are no longer hard-wired. They too can be exchanged via LoadModule. The default MPM in 2.3 is the event MPM.

The maximum number of workers, be they pre-forked child processes or threads within a process, is an indication of how many requests your server can manage concurrently. It is merely a rough estimate because the kernel can queue connection attempts for your web server. When your site becomes busy and the maximum number of workers is running, the machine doesn't hit a hard limit beyond which clients will be denied access. However, once requests start backing up, system performance is likely to degrade.

MaxClients

The `MaxClients` directive in your Apache httpd configuration file specifies the maximum number of workers your server can create. It has two related directives,
`MinSpareServers` and `MaxSpareServers`, which specify the number of workers Apache keeps waiting in the wings ready to serve requests. The absolute maximum number of processes is configurable through the `ServerLimit` directive.

Spinning Threads

For the prefork MPM of the above directives are all there is to determining the process limit. However, if you are running a threaded MPM the situation is a little more complicated. Threaded MPMs support the `ThreadsPerChild` directive1 . Apache requires that `MaxClients` is evenly divisible by `ThreadsPerChild`. If you set either directive to a number that doesn't meet this requirement, Apache will send a message of complaint to the error log and adjust the `ThreadsPerChild` value downwards until it is an even factor of `MaxClients`.

Sizing MaxClients

Optimally, the maximum number of processes should be set so that all the memory on your system is used, but no more. If your system gets so overloaded that it needs to heavily swap core memory out to disk, performance will degrade quickly. The formula for determining `MaxClients` is fairly simple:

```
              total RAM  RAM for OS  RAM for external programs
MaxClients = ------------------------------------------------------
                           RAM per httpd process
```

The various amounts of memory allocated for the OS, external programs and the httpd processes is best determined by observation: use the top and free commands described above to determine the memory footprint of the OS without the web server running. You can also determine the footprint of a typical web server process from top: most top implementations have a Resident Size (RSS) column and a Shared Memory column.

The difference between these two is the amount of memory per-process. The shared segment really exists only once and is used for the code and libraries loaded and the dynamic inter-process tally, or 'scoreboard,' that Apache keeps. How much memory each process takes for itself depends heavily on the number and kind of modules you use. The best approach to use in determining this need is to generate a typical test load against your web site and see how large the httpd processes become.

The RAM for external programs parameter is intended mostly for CGI programs and scripts that run outside the web server process. However, if you have a Java virtual machine running Tomcat on the same box it will need a significant amount of memory as well. The above assessment should give you an idea how far you can push `MaxClients`, but it is not an exact science. When in doubt, be conservative and use a low `MaxClients` value. The Linux kernel will put extra memory to good use for caching disk access. On Solaris you need enough available real RAM memory to create any process. If no real memory is available, httpd will start writing 'No space left on device' messages to the error log and be unable to fork additional child processes, so a higher `MaxClients`
value may actually be a disadvantage.

Selecting your MPM

The prime reason for selecting a threaded MPM is that threads consume fewer system resources than processes, and it takes less effort for the system to switch between threads. This is more true for some operating systems than for others. On systems like Solaris and AIX, manipulating processes is relatively expensive in terms of system resources. On these systems, running a threaded MPM makes sense. On Linux, the threading implementation actually uses one process for each thread. Linux processes are relatively lightweight, but it means that a threaded MPM offers less of a performance advantage than in other environments.

Running a threaded MPM can cause stability problems in some situations For instance, should a child process of a preforked MPM crash, at most one client connection is affected. However, if a threaded child crashes, all the threads in that process disappear, which means all the clients currently being served by that process will see their connection aborted. Additionally, there may be so-called "thread-safety" issues, especially with third-party libraries. In threaded applications, threads may access the same variables indiscriminately, not knowing whether a variable may have been changed by another thread.

This has been a sore point within the PHP community. The PHP processor heavily relies on third-party libraries and cannot guarantee that all of these are thread-safe. The good news is that if you are running Apache on Linux, you can run PHP in the preforked MPM without fear of losing too much performance relative to the threaded option.

Spinning Locks

Apache httpd maintains an inter-process lock around its network listener. For all practical purposes, this means that only one httpd child process can receive a request at any given time. The other processes are either servicing requests already received or are 'camping out' on the lock, waiting for the network listener to become available. This process is best visualized as a revolving door, with only one process allowed in the door at any time. On a heavily loaded web server with requests arriving constantly, the door spins quickly and requests are accepted at a steady rate. On a lightly loaded web server, the process that currently "holds" the lock may have to stay in the door for a while, during which all the other processes sit idle, waiting to acquire the lock. At this time, the parent process may decide to terminate some children based on its `MaxSpareServers` directive.

The Thundering Herd

The function of the 'accept mutex' (as this inter-process lock is called) is to keep request reception moving along in an orderly fashion. If the lock is absent, the server may exhibit the Thundering Herd syndrome.

Consider an American Football team poised on the line of scrimmage. If the football players were Apache processes all team members would go for the ball simultaneously at the snap. One process would get it, and all the others would have to lumber back to the line for the next snap. In this metaphor, the accept mutex acts as the quarterback, delivering the connection "ball" to the appropriate player process.

Moving this much information around is obviously a lot of work, and, like a smart person, a smart web server tries to avoid it whenever possible. Hence the revolving door construction. In recent years, many operating systems, including Linux and Solaris, have put code in place to prevent the Thundering Herd syndrome. Apache recognizes this and if you run with just one network listener, meaning one virtual host or just the main server, Apache will refrain from using an accept mutex. If you run with multiple listeners (for instance because you have a virtual host serving SSL requests), it will activate the accept mutex to avoid internal conflicts.

You can manipulate the accept mutex with the `AcceptMutex` directive. Besides turning the accept mutex off, you can select the locking mechanism. Common locking mechanisms include fcntl, System V Semaphores and pthread locking. Not all are available on every platform, and their availability also depends on compile-time settings. The various locking mechanisms may place specific demands on system resources: manipulate them with care.

There is no compelling reason to disable the accept mutex. Apache automatically recognizes the single listener situation described above and knows if it is safe to run without mutex on your platform.

Tuning the Operating System

People often look for the 'magic tune-up' that will make their system perform four times as fast by tweaking just one little setting. The truth is, present-day UNIX derivatives are pretty well adjusted straight out of the box and there is not a lot that needs to be done to make them perform optimally. However, there are a few things that an administrator can do to improve performance.

RAM and Swap Space

The usual mantra regarding RAM is "more is better". As discussed above, unused RAM is put to good use as file system cache. The Apache processes get bigger if you load more modules, especially if you use modules that generate dynamic page content within the processes, like PHP and mod_perl. A large configuration file-with many virtual hosts-also tends to inflate the process footprint. Having ample RAM allows you to run Apache with more child processes, which allows the server to process more concurrent requests.

While the various platforms treat their virtual memory in different ways, it is never a good idea to run with less disk-based swap space than RAM. The virtual memory system is designed to provide a fallback for RAM, but when you don't have disk space available and run out of swappable memory, your machine grinds to a halt. This can crash your box, requiring a physical reboot for which your hosting facility may charge you.

Also, such an outage naturally occurs when you least want it: when the world has found your website and is beating a path to your door. If you have enough disk-based swap space available and the machine gets overloaded, it may get very, very slow as the system needs to swap memory pages to disk and back, but when the load decreases the system should recover. Remember, you still have `MaxClients` to keep things in hand.

Most unix-like operating systems use designated disk partitions for swap space. When a system starts up it finds all swap partitions on the disk(s), by partition type or because they are listed in the file `/etc/fstab`, and automatically enables them. When adding a disk or installing the operating system, be sure to allocate enough swap space to accommodate eventual RAM upgrades. Reassigning disk space on a running system is a cumbersome process.

Plan for available hard drive swap space of at least twice your amount of RAM, perhaps up to four times in situations with frequent peaking loads. Remember to adjust this configuration whenever you upgrade RAM on your system. In a pinch, you can use a regular file as swap space. For instructions on how to do this, see the manual pages for the `mkswap` and `swapon` or `swap` programs.

ulimit: Files and Processes

Given a machine with plenty of RAM and processor capacity, you can run hundreds of Apache processes if necessary. . . and if your kernel allows it.

Consider a situation in which several hundred web servers are running; if some of these need to spawn CGI processes, the maximum number of processes would occur quickly.

However, you can change this limit with the command

```
ulimit [-H|-S] -u [newvalue]
```

This must be changed before starting the server, since the new value will only be available to the current shell and programs started from it. In newer Linux kernels the default has been raised to 2048. On FreeBSD, the number seems to be the rather unusual 513. In the default user shell on this system, `csh` the equivalent is `limit` and works analogous the the Bourne-like `ulimit`:

```
limit [-h] maxproc [newvalue]
```

Similarly, the kernel may limit the number of open files per process. This is generally not a problem for pre-forked servers, which just handle one request at a time per process. Threaded servers, however, serve many requests per process and much more easily run out of available file descriptors. You can increase the maximum number of open files per process by running the

```
ulimit -n [newvalue]
```

command. Once again, this must be done prior to starting Apache.

Setting User Limits on System Startup

Under Linux, you can set the ulimit parameters on bootup by editing the `/etc/security/limits.conf` file. This file allows you to set soft and hard limits on a per-user or per-group basis; the file contains commentary explaining the options. To enable this, make sure that the file `/etc/pam.d/login` contains the line

```
session required /lib/security/pam_limits.so
```

All items can have a 'soft' and a 'hard' limit: the first is the default setting and the second the maximum value for that item.

In FreeBSD's `/etc/login.conf` these resources can be limited or extended system wide, analogously to `limits.conf`. 'Soft' limits can be specified with `-cur` and 'hard' limits with `-max`.

Solaris has a similar mechanism for manipulating limit values at boot time:
In `/etc/system` you can set kernel tunables valid for the entire system at boot time. These are the same tunables that can be set with the `mdb` kernel debugger during run time. The soft and hard limit corresponding to ulimit -u can be set via:

```
set rlim_fd_max=65536
set rlim_fd_cur=2048
```

Solaris calculates the maximum number of allowed processes per user (`maxuprc`) based on the total amount available memory on the system (`maxusers`) . You can review the numbers with

```
sysdef -i | grep maximum
```

but it is not recommended to change them.

Turn Off Unused Services and Modules

Many UNIX and Linux distributions come with a slew of services turned on by default. You probably need few of them. For example, your web server does not need to be running sendmail, nor is it likely to be an NFS server, etc. Turn them off.

On Red Hat Linux, the chkconfig tool will help you do this from the command line. On Solaris systems `svcs` and `svcadm` will show which services are enabled and disable them respectively.

In a similar fashion, cast a critical eye on the Apache modules you load. Most binary distributions of Apache httpd, and pre-installed versions that come with Linux distributions, have their modules enabled through the `LoadModule` directive.

Unused modules may be culled: if you don't rely on their functionality and configuration directives, you can turn them off by commenting out the corresponding `LoadModule`
lines. Read the documentation on each module's functionality before deciding whether to keep it enabled. While the performance overhead of an unused module is small, it's also unnecessary.

Caching Content

Requests for dynamically generated content usually take significantly more resources than requests for static content. Static content consists of simple filespages, images, etc.-on disk that are very efficiently served. Many operating systems also automatically cache the contents of frequently accessed files in memory.

Processing dynamic requests, on the contrary, can be much more involved. Running CGI scripts, handing off requests to an external application server and accessing database content can introduce significant latency and processing load to a busy web server. Under many circumstances, performance can be improved by turning popular dynamic requests into static requests. In this section, two approaches to this will be discussed.

Making Popular Pages Static

By pre-rendering the response pages for the most popular queries in your application, you can gain a significant performance improvement without giving up the flexibility of dynamically generated content. For instance, if your application is a flower delivery service, you would probably want to pre-render your catalog pages for red roses during the weeks leading up to Valentine's Day. When the user searches for red roses, they are served the pre-rendered page. Queries for, say, yellow roses will be generated directly from the database. The mod_rewrite module included with Apache is a great tool to implement these substitutions.

Example: A Statically Rendered Blog

'we should provide a more useful example here. One showing how to make Wordpress or Drupal suck less.'

Blosxom is a lightweight web log package that runs as a CGI. It is written in Perl and uses plain text files for entry input. Besides running as CGI, Blosxom can be run from the command line to pre-render blog pages. Pre-rendering pages to static HTML can yield a significant performance boost in the event that large numbers of people actually start reading your blog.

To run blosxom for static page generation, edit the CGI script according to the documentation. Set the $static dir variable to the `DocumentRoot` of the web server, and run the script from the command line as follows:

```
$ perl blosxom.cgi -password='whateveryourpassword'
```

This can be run periodically from Cron, after you upload content, etc. To make Apache substitute the statically rendered pages for the dynamic content, we' ll use mod_rewrite. This module is included with the Apache source code, but is not compiled by default. It can be built with the server by passing the option `--enable-rewrite[=shared]` to the configure command. Many binary distributions of Apache come with mod_rewrite included. The following is an example of an Apache virtual host that takes advantage of pre-rendered blog pages:

```
Listen *:8001
<VirtualHost *:8001>
  ServerName blog.sandla.org:8001
  ServerAdmin sander@temme.net
  DocumentRoot "/home/sctemme/inst/blog/httpd/htdocs"
  <Directory "/home/sctemme/inst/blog/httpd/htdocs">
    Options +Indexes
    Order allow,deny
    Allow from all
    RewriteEngine on
    RewriteCond %{REQUEST_FILENAME} !-f
    RewriteCond %{REQUEST_FILENAME} !-d
    RewriteRule ^(.*)$ /cgi-bin/blosxom.cgi/$1 [L,QSA]
  </Directory>
  RewriteLog /home/sctemme/inst/blog/httpd/logs/rewrite_log
  RewriteLogLevel 9
  ErrorLog /home/sctemme/inst/blog/httpd/logs/error_log
  LogLevel debug
  CustomLog /home/sctemme/inst/blog/httpd/logs/access_log common
  ScriptAlias /cgi-bin/ /home/sctemme/inst/blog/bin/
  <Directory "/home/sctemme/inst/blog/bin">
    Options +ExecCGI
    Order allow,deny
    Allow from all
  </Directory>
</VirtualHost>
```

The `RewriteCond` and `RewriteRule` directives say that, if the requested resource does not exist as a file or a directory, its path is passed to the Blosxom CGI for rendering. Blosxom uses Path Info to specify blog entries and index pages, so this means that if a particular path under Blosxom exists as a static file in the file system, the file is served instead. Any request that isn't pre-
rendered is served by the CGI. This means that individual entries, which show the comments, are always served by the CGI which in turn means that your comment spam is always visible. This configuration also hides the Blosxom CGI from the user-visible URL in their Location bar. mod_rewrite is a fantastically powerful and versatile module: investigate it to arrive at a configuration that is best for your situation.

Caching Content With mod_cache

The mod_cache module provides intelligent caching of HTTP responses: it is aware of the expiration timing and content requirements that are part of the HTTP specification. The mod_cache module caches URL response content. If content sent to the client is considered cacheable, it is saved to disk. Subsequent requests for that URL will be served directly from the cache. The provider module for mod_cache, mod_disk_cache, determines how the cached content is stored on disk. Most server systems will have more disk available than memory, and it's good to note that some operating system kernels cache frequently accessed disk content transparently in memory, so replicating this in the server is not very useful.

To enable efficient content caching and avoid presenting the user with stale or invalid content, the application that generates the actual content has to send the correct response headers. Without headers like `Etag:`, `Last-Modified:` or `Expires:`, mod_cache can not make the right decision on whether to cache the content, serve it from cache or leave it alone. When testing content caching, you may find that you need to modify your application or, if this is impossible, selectively disable caching for URLs that cause problems. The mod_cache modules are not compiled by default, but can be enabled by passing the option `--enable-cache[=shared]` to the configure script. If you use a binary distribution of Apache httpd, or it came with your port or package collection, it may have mod_cache already included.

Example: wiki.apache.org

'Is this still the case? Maybe we should give a better example here too.

The Apache Software Foundation Wiki is served by MoinMoin. MoinMoin is written in Python and runs as a CGI. To date, any attempts to run it under mod_python has been unsuccessful. The CGI proved to place an untenably high load on the server machine, especially when the Wiki was being indexed by search engines like Google. To lighten the load on the server machine, the Apache Infrastructure team turned to mod_cache. It turned out MoinMoin needed a small patch to ensure proper behavior behind the caching server:
certain requests can never be cached and the corresponding Python modules were patched to send the proper HTTP response headers. After this modification, the cache in front of the Wiki was enabled with the following configuration snippet in `httpd.conf`:

```
CacheRoot /raid1/cacheroot
CacheEnable disk /
# A page modified 100 minutes ago will expire in 10 minutes
CacheLastModifiedFactor .1
# Always check again after 6 hours
CacheMaxExpire 21600
```

This configuration will try to cache any and all content within its virtual host. It will never cache content for more than six hours (the `CacheMaxExpire` directive). If no `Expires:` header is present in the response, mod_cache will compute an expiration period from the `Last-Modified:` header. The computation using `CacheLastModifiedFactor` is based on the assumption that if a page was recently modified, it is likely to change again in the near future and will have to be re-cached.

Do note that it can pay off to *disable* the `ETag:` header: For files smaller than 1k the server has to calculate the checksum (usually MD5) and then send out a `304 Not Modified` response, which will take waste some CPU and still saturate the same amount of network resources for the transfer (one TCP packet). For resources larger than 1k it might prove CPU expensive to calculate the header for each request. Unfortunately there does currently not exist a way to cache these headers.

```
<FilesMatch \.(jpe?g|png|gif|js|css|x?html|xml)>
  FilesETag None
</FilesMatch>
```

This will disable the generation of the `ETag:` header for most static resources. The server does not calculate these headers for dynamic resources.

Further Considerations

Armed with the knowledge of how to tune a sytem to deliver the desired the performance, we will soon discover that *one* system might prove a bottleneck. How to make a system fit for growth, or how to put a number of systems into tune will be discussed in PerformanceScalingOut.