Apache Performance Tuning
Part One: Scaling Up

Sander Temme
sander@temme.net

June 30, 2006

Abstract

The Apache HTTP Server powers some of the busiest sites in the world, so when your web site takes off and becomes popular, you're in good hands when you use Apache. This session covers tuning the Apache HTTP Server for performance. We will discuss server performance monitoring and benchmarking. Then we will cover configuration options that affect performance, system sizing and operating system tuning. Finally, we will discuss caching dynamic content with *mod_cache*.

1 Introduction

The Performance Tuning page in the Apache 1.3 documentation says[1]:

> "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 paper and the ApacheCon session it accompanies, several aspects of web server performance will be discussed.

[1] http://httpd.apache.org/docs/misc/perf-tuning.html

1.1 What Will and Will Not Be Discussed

The session will focus on easily accessible configuration and tuning options for Apache 1.3 and 2 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 configuration file.

2 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.

2.1 Monitoring Tools

2.1.1 top

The *top* tool ships with Linux and FreeBSD, and can be downloaded for Solaris[2]. 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. An example *top* display is shown in Figure 1.

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 Section 3.1.3. Top is, however, an interactive tool and running it continuously has few if any advantages.

2.1.2 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

[2] http://www.sunfreeware.com/

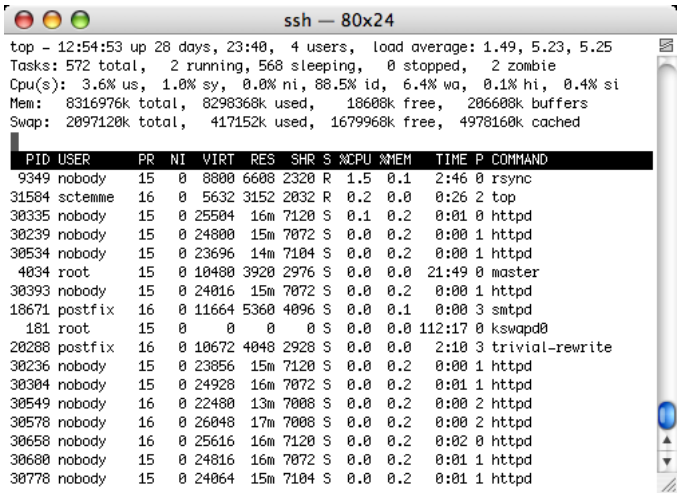

Figure 1: The *top* tool offers a running display of system performance statistics and individual process data

described in the paragraph *'Sizing MaxClients'* on page 8. 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
```

2.1.3 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.

2.1.4 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 [6].

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[3], 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 Metrics[4] that plans to bring to market a multiplatform monitoring tool built on the same principles as *SE Toolkit*, written in Java.

2.1.5 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 Apache and the rest of the system. The Apache Software Foundation uses *mod_status* on its own web site[5]. 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.

2.2 Web Server Log Files

Apache creates log files, which can yield a lot of information about server performance.

[3] http://www.sunfreeware.com/setoolkit.htm
[4] http://www.setoolkit.com/
[5] http://www.apache.org/server-status

Figure 2: The *SE Toolkit* gathers statistics from the Solaris OS.

2.2.1 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. Indeed, any error encountered by Apache will appear in this log, so it is good practice to review it frequently.

The Error Log is configured through the `ErrorLog` and `LogLevel` configuration directives. The *error_log* of Apache'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, the latter will predominantly be worm attacks and exploit attempts.

The `LogLevel` directive determines the level of detail included in the logs. There are eight log levels, described in Table 1. 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.

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

Table 1: Apache log levels

2.2.2 Access Log

Apache keeps track of every request it services in its *access_log* file. In addition to the time and nature of a request, Apache 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 Apache manual[6]. 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[7] and Webalizer[8]. 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 Table 2.

2.3 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_load*[9]. This program works with a URL file and can be compiled with SSL support.

[6] http://httpd.apache.org/docs/mod/mod_log_config.html# formats
[7] http://www.analog.cx/
[8] http://www.mrunix.net/webalizer/
[9] http://www.acme.com/software/http_load/

```
172.16.2.91 - - [16/Nov/2003:15:23:27 -0800] "GET /~sctemme/ HTTP/1.1"
172.16.2.200 1446 91 - - [16/Nov/2003:15:23:28 -0800] "GET
172.16.2./~sctemme/css/doc.css HTTP/1.1" 200 167 91 - -
172.16.2.[16/Nov/2003:15:23:28 -0800] "GET /~sctemme/css/menu.css
172.16.2.HTTP/1.1" 200 623 91 - - [16/Nov/2003:15:23:29 -0800] "GET
172.16.2./favicon.ico HTTP/1.1" 404 283 91 - - [16/Nov/2003:15:23:34
172.16.2.-0800] "GET /~sctemme/index.html HTTP/1.1" 200 1446
```

Field	Content	Explanation
Client IP	172.16.2.91	IP address where the request originated
RFC 1413 ident	-	Remote user identity as reported by their identd
username	-	Remote username as authenticated by Apache
timestamp	[16/Nov/2003:15:23:29 -0800]	Date and time of request
Request	"GET /favicon.ico HTTP/1.1"	Request line
Status Code	404	Response code
Content Bytes	283	Bytes transferred w/o headers

Table 2: Common Log Format fields explained

- The Apache Software Foundation offers a tool named *flood*[10]. Flood is a fairly sophisticated program that is configured through an XML file.
- Finally, *JMeter*[11], 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.

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.

3 Configuring for Performance

3.1 Apache Configuration

The Apache 1.3 httpd is 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. Apache 2 introduced the concept of the Multi-Processing Module (MPM). Developers can write MPMs to suit the process- or threading-architecture 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

[10] http://httpd.apache.org/test/flood/
[11] http://jakarta.apache.org/jmeter/

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.

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.

3.1.1 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 hard coded into Apache 1.3 as the parameter HARD_SERVER_LIMIT: in order to change it you'd have to recompile the server. Fortunately, most distributors have raised this limit well beyond the default of 256. In Apache 2.0, this limit is configurable through the ServerLimit directive.

3.1.2 Spinning Threads

For Apache 1.3, or the prefork MPM of Apache 2.0, 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 directive[12]. 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.

3.1.3 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:

$$MaxClients = \frac{total\ RAM - RAM\ for\ OS - RAM\ for\ external\ programs}{RAM\ per\ httpd\ process}$$

The various amounts of memory allocated for the OS, external programs and the httpd processes is best determined by observation: use the top and

[12] http://httpd.apache.org/docs-2.0/mod/mpm_common.html #threadsperchild

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, Apache 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.

3.1.4 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 indiscriminantly, not knowing whether a variable may have been changed by another thread.

This has been a sore point within the PHP community[13]. The PHP processor heavily relies on third-party libraries and cannot guarantee that all of these are

[13] http://www.php.net/

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.

3.1.5 Spinning Locks

Apache 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, durin 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.

3.1.6 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[14] 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.

[14] Any resemblance to the college in West Virginia is purely coincidental.

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[15].

3.2 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.

3.2.1 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. Finally, Apache 2.0 processes tend to have larger footprints than those of Apache 1.3. 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 accomodate 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. Re-

[15] To find out, run httpd with the -V flag and see if -D SINGLE_LISTEN_UNSERIALIZED_ACCEPT appears in the output. You can see which locking mechanisms are available by running httpd -L, and browsing the output for the AcceptMutex directive.

member 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.

3.2.2 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. The Linux 2.2 kernel series by default limited the number of processes a user can run to 256, which is a fairly small number in computational circles.

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 -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. There does not seem to be a process limit on Solaris. On FreeBSD, the number seems to be the rather unusual 513. Since the default user shell on this system is csh and ulimit is a built-in command in Bourne-like shells, you need to start a Bourne shell or bash to see and set this value, and then start the web server from this shell.

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.

3.2.3 Setting User Limits on Linux 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.

Solaris does not seem to have a similar mechanism for manipulating limit values at boot time: you will have to set them in your startup script(s).

3.2.4 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, my approach is to inspect the /etc/rc[123].d directories and to change the first character of the name of startup scripts I don't want to start automatically from S to s. Since the Solaris file system is case sensitive, this disables services without actually altering them so they become unrecognizable.

Tip

Thus, S88sendmail becomes s88sendmail. This way, the init process will pass them over but it's still evident to other sysadmins that they were once active. While Solaris transitions through subsequent run levels on startup, the Linux initialization just executes all the scripts in the default run level directory. The default run level for a Linux web server should be 3: you don't need to run an X-Windows desktop on a web server so level 5 should not be necessary.

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

A notable exception is the Apache httpd on Cobalt Raq servers, which has *mod_perl* compiled statically to run the GUI–despite the fact that the GUI Apache is running as an entirely different process from the one doing the actual serving. You cannot disable this instance of *mod_perl*. Other modules, however, may be culled: if you don't use their functionality and configuration directives, you can turn them off by commenting out the corresponding LoadModule lines. Read the documentation[16] 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.

4 Caching Content

Requests for dynamically generated content usually take significantly more resources than requests for static content. Static content consists of simple files–pages, images, etc.–on disk that are very efficiently served. On platforms that support it, Apache uses the `sendfile(2)` system call to instruct the operating system kernel to transfer the contents of requested files directly to network sockets. The server does not need to read or inspect the contents of the files which makes this a very efficient operation. 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

[16] http://httpd.apache.org/docs/2.2/ for Apache 2, and http://httpd.apache.org/docs/1.3/ for Apache 1.3

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.

4.1 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.

4.1.1 Example: A Statically Rendered Blog

Blosxom[17] 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 [9]. 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:

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

[17] http://www.blosxom.com/

```
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>
```

Figure 3: Configuration snippet for a blog with potentially statically rendered pages

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.

4.2 Caching Content With mod_cache

As described in [8], *mod_cache* is no longer considered *experimental* in httpd 2.2 and is now included in the base distribution. 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_mem_cache* or *mod_disk_cache*, determines whether the cached content is stored on disk or in memory. 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.

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, or it came with your port or package collection, it may have *mod_cache* already included.

4.2.1 Example: wiki.apache.org

The Apache Software Foundation Wiki is served by MoinMoin[18]. 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 behaviour 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:

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

[18] http://moinmoin.wikiwikiweb.de/

```
<IfModule mod_cache.c>
  <IfModule mod_disk_cache.c>

  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
 </IfModule>

</IfModule>
```

Figure 4: Configuration snippet to enable dynamic content caching on wiki.apache.org

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. Please see the *mod_cache* documentation [8] for more information on using this module.

5 Conclusion

As you can see, there are a number of effective ways to tune an Apache system to perform better. Armed with a basic knowledge of the system's original performance and its desired memory footprint, an administrator can make small changes which may result in dramatic improvements. Setting system variables correctly, turning off unnecessary processes, and distributing the load can all be excellent tools for making your less than state of the art computer more fleet of foot.

References

[1] Ryan B. Bloom, *Apache Server 2.0: The Complete Reference*, 2002 McGraw Hill Osborne; ISBN 0-07-222344-8

[2] Ben Laurie and Peter Laurie, *Apache: The Definitive Guide (3rd Edition)*, 2002 O'Reilly & Associates; ISBN 0596002033

[3] Patrick Killelea, *Web Performance Tuning, 2nd Edition*, 2002 O'Reilly & Associates; ISBN 0-596-00172-X

[4] http://httpd.apache.org/docs/2.2/misc/perf-tuning.html

[5] http://httpd.apache.org/docs/1.3/misc/perf-tuning.html

[6] Adrian Cockcroft, Richard Pettit and Sun Microsystems Press, *Sun Performance and Tuning: Java and the Internet (2nd Edition)*, 1998 Prentice Hall PTR; ISBN 0130952494

[7] Ken Coar and Rich Bowen, *Apache Cookbook*, 2003 O'Reilly & Associates; ISBN 0596001916

[8] http://httpd.apache.org/docs/2.2/caching.html

[9] http://www.blosxom.com/documentation/users/configure/static.html