Tuning the Apache Server: Getting the Most Out of Your httpd

Chris Uriarte <chrisjur@cju.com>

The first line of the *Apache Perfomance Notes* (http://www.apache.org/docs/misc/perf-tuning.html) says it all: *"Apache is a general web server, which is designed to be correct first, and fast second"*. Far too often have I come across busy Apache web environments that to use "out of the box" configurations. I usually ask this question to system admins and MIS managers:

Question: "When you install a copy of Solaris or Windows NT Server, do you assume that it's secure and tweaked for optimum performance right out of the box?".

Typical Answer: "Of course not."

Well, then why should Apache be any different? In an environment that revolves around the web server, like a corporate intranet or a web-hosting network, administrators should give as much attention to tuning their web server as they do the operating system it runs on. In this paper, we'll examine some of the tricks and tips you can take advantage of to get the most out of Apache.

Before we get into details, it may be helpful to give a simple overview of how Apache works. Apache's startup method is know as the "pre-fork" model, since it spawns, or forks, a pool of child processes when it starts up (the term fork is taken from the UNIX system call, fork(), that is used to create identical child processes of a specific program). If the `StartServers` directive in your httpd.conf file is set to 10, Apache will create 10 child processes upon startup. Each child process is used to handle simultaneous client connections to your website. If you have more simultaneous connections than child processes running, Apache will spawn additional child servers as needed, up to a limit that you can specify via the `MaxClients` directive in your httpd.conf. It is important to realize that the more "forking" a server has to do, the greater performance hit the server takes, which is why we create spare servers, or children, upon startup. It's also important to realize that the more child servers you have activated, the more memory Apache eats up. We're going to address how this model requires some tweaking to get the maximum performance out of your Apache server.

Start With Your Hardware

It's no secret that if you want your applications on your workstation to run faster and more efficiently, the quickest and most effective solution is to upgrade your RAM. People are often wrongly concerned with the speed of their processor, rather than the amount of RAM they have. After all, if you have a Pentium III processor running at 500 MHz, but only have 32 Megs or RAM, your applications are going to slow down very quickly. The same concept applies to Apache – if you don't have enough RAM, its performance is going to quickly take a nosedive. When your UNIX system runs out of physical RAM, it starts to utilize the system's swap space. The more and more swap space your system uses, the slower Apache will perform. Your overall goal should be to eliminate swapping all together. You can accomplish this in a couple of ways.

First, and most obvious of all, make sure you have enough RAM in your system. Determining how much RAM you require may take a little "guestimation". You can start by finding out just how much memory each separate Apache child process requires. You can determine this by using a utility like *top*. Now, determine the amount of `MaxClients` that are set in your httpd.conf file. The `MaxClients` directive sets the maximum amount of servers that Apache can run at one time. If you receive enough hits that require `MaxClients` servers to be spawned, all additional clients will effectively be locked out of your site until a server process is freed up. So, if the number of `MaxClients` is set to 50 and each httpd child requires 2 MB or memory, they you should have at least 100MB of memory available to Apache. Running CGI applications also requires additional memory each time the CGI is executed. So if you're running CGIs, it's probably a good idea to make sure you have an ample amount of memory, above and beyond what the core Apache server requires.

Make Sure Your MaxClients Is Not Set Too High –
Upgrade or Distribute if You Run Out of Breathing Room

If the `MaxClients` directive in your httpd.conf is set too high, Apache may spawn too many child servers. If Apache spawns too many children, your sever may run out of RAM and start to swap. If you don't have a terribly busy website, Apache's default setting of 150 is fine. However if you set your `MaxClients` to the maximum amount of servers that Apache can spawn without swapping and you reach that limit, it's an indicator that it's time to upgrade your RAM or distribute your load across multiple servers.

On Larger Sites, Tweak some of the Often-Ignored Server Settings

O'Reilly's *Apache: The Definitive Guide* and Apache's *Performance Tuning Notes* both note how `MinSpareServers, MaxSpareServers` and the `StartServers` settings used to have a drastic effect on Apache performance before version 1.3. However, since version 1.3 was released, we don't need to be too concerned with these values. Ben and Peter Laurie note that websites which handle 1 million hits a day work well when `MaxSpareServers` set to 64, `MinSpareServers` set to 32 and StartServers set somewhere in between those two values. If you are running a smaller website, you need not be too concerned with these values.

Get Rid of .htaccess Files

If you have something like **`AllowOverride all`** in your configuration files, Apache will look for a *.htaccess* file every time a request is made. Use **`AllowOverride None`** in your configuration file and specify your .htaccess directives in your server configuration files, instead. You can add something like this to your httpd.conf file instead of the usual .htaccess file:

```
<Directory /members>
      AuthType Basic
      AuthName members-area
      Require valid-user
      AuthUserFiles /usr/local/userfile
</Directory>
```

Turn off DNS Hostname Lookups

With the 1.3 release of Apache, the `HostNameLookups` directive is now turned off by default. If this directive is turned on, Apache must do a reverse DNS lookup for every client request. This can take a great amount of time, especially on networks with slow DNS servers. The only purpose this feature serves is to record a client's hostname in your log files, rather than a simple IP address. But if you are really interested in who's visiting your website, even the free log analysis tools available on the web will do the hostname resolution for you when it comes time to analyze your log files.

Don't Use FollowSymLinks and SymLinksIfOwnerMatch

If you enable the `FollowSymLinks` and `SymLinksIfOwnerMatch` directives, Apache will have to do some additional file checking each time a request is made. Many administrators are hesitant to use these directives anyway, since they may open security holes by allowing users to link to files of the web document tree (.i.e. linking to /etc/passwd).

Make Your Negotiation More Specific

The `DirectoryIndex` directive in your configuration files sets the name of the default file that the server will return for each request to a directory (i.e. a request to http://server.com may return index.html from the root web directory). You have the option of adding a line like this to your config files:

```
DirectoryIndex    index
```

This is know as a "wildcard" directive. When you specify a wildcard directive, Apache has to perform content negotiation on the index file it finds. Instead of using wildcard directives you should specify a list of specific file names like:

```
DirectoryIndex    index.html index.htm index.cgi index.shtml
```

To squeeze the last bit of performance out of your system, place your most commonly used index file first in the list and your least used index file last.

Trash the XbitHack
The XbitHack is an obsolete facility that only exists for backward compatibility. Using this hack, a file is parsed for Server Side Includes if its group execution bit is set. If you're still using this hack, get rid of it because it requires additional file checking to be executed on each request.

Only use the Modules You Need – Don't Use DSO Unless You Have To
The more modules you compile into Apache, the more memory each server requires. With the release of version 1.3, you can compile in Dynamic Shared Object support, which allows you to add modules without re-compiling the server. DSO support requires significant overhead and increases the amount of memory each child server requires. Here's a list of some modules that you can get rid of:

- If you are not familiar with reading the Apache status report and don't think you would use it for debugging or performance evaluations, you can get rid of `mod_status` and `mod_info`.

- If you don't want to allow automatic spelling correction, get rid of `mod_speling` (no, "speling" is not misspelled). Chances are you've lived without this feature most of your life on the web, and you can do without it.

- If you don't need proxy support, don't use compile in `libproxy.a`. If you are just doing straight serving of web pages, or if you don't know what a proxy is, chances are you don't need this feature.

- `mod_auth_anon` enables anonymous ftp-like authentication and is not widely used. Chances are you don't need this feature.

- `mod_auth_dbm` and `mod_auth_db` let you use Berkeley DB files to hold user authentication information. If you don't plan to use these types of files or if you system does not support DBM files, you won't need these modules.

- `mod_digest` implements Digest authentication, which is more secure than Basic authentication, where usernames and passwords are sent to the server in clear text. The concept behind Digest authentication is great, but unfortunately, many versions of popular browsers do not support it. If you are in an environment where you can control the browser being used (corporate intranet?), you may want to explore using Digest authentication. However, it is pretty useless when serving pages over the web. If you'd like to check if your browser has Digest support, check out http://digest-test.agranat.com/test_list.html.

- Without getting into too much detail, `mod_cern_meta` and `mod_example` are often not used. Chances are, you won't need them.

- `mod_usertrack` used to be `mod_cookies.` This module uses Netscape cookies to track click trails on your site. You may find this option interesting if you are interested in tracking a user's journey throughout your website. If you're not interested in user tracking, then this module is not required. It's important to note that you *do not* need to compile in this module if you wish to use HTTP Cookies throughout your site.

- `mod_unique_id` does not work on all UNIX distributions, but may be helpful if you're using a web application that needs to track user sessions. If this does not sound appealing to you, you don't need to compile it in.

- `mod_so` is what enables DSO support. If you are not going to be experimenting with modules that need to be added or removed on the fly, you don't need to enable DSO support.

- `mod_headers` let's you set your own custom HTTP headers in your configuration files. If you're not interested in this feature, don't compile it in.

The Big One: Stop Using CGI - Find a Better way

If your site relies on a lot of CGI applications, you're server is taking a big performance hit. Every time an external CGI program is called, the server has to fork another process. When using Perl for your CGI scripts, your Perl script and its associated modules have to be compiled and re-compiled every time the CGI program is called. If your site needs to connect to a database, your CGI application must log into the database and create a TCP session *every time* the CGI program is called. The solution is to crack right into the Apache API by using programs written in C, or programs written in Perl using mod_perl (http://perl.apache.org). You can also use Java servlets in conjunction with Apache Jserve (http://java.apache.org). Using a language like PHP also helps improve performance because the PHP engine is built right into the Apache binary (although PHP requires line-by-line parsing of every .php file). You can also get a performance boost by getting rid of server-parsed html files that use Server Side Includes (SSI). So how much boost in performance can you get by using an program that takes advantage of the Apache API? Well, some people have reported a performance boost of 400% - yes 400%. Actually, in our own tests using a script that utilizes the Perl DBI database interface package, we saw some performance boosts as high as 600%

All in all, you'll find that by using some of these easy-to-implement tricks, you can get the most out of Apache's performance, minimize the amount of RAM Apache eats and drastically improve the performance of your web applications.