Pippilotta*, www.photocase.com

The Cherokee and Lighttpd alternative web servers

FEATHERWEIGHTS

Apache has ruled the web since the mid-90s, but not all users are happy with it. Recent competitors Cherokee and Lighttpd offer an uncomplicated alternative for users who are looking for something light.

BY OLIVER FROMMEL

Apache is so predominant among Unix-style web servers that most people wouldn't dream of looking for alternatives. On the other hand, the Apache server carries much ballast, some of which dates back to its previous life as the NCSA-HTTPD.

THE AUTHOR

For several years Oliver was a sysop and programmer at Ars Electronica Center in Linz/Austria. After finishing his studies in Philosophy, Linguistics and Computer Science he became an editor for the Bavarian Broadcasting Corporation. He is responsible for special editions of Linux Magazine Germany.

For example, the Apache developers kept the multi-process model with a fixed number of pre-launched server instances until version 2. Now there is a variant that uses multiple threads to process requests; then again, this new construction has added to the existing codebase, which can lead to inefficiencies.

In this article, I examine a pair of promising Apache alternatives.

Cherokee

Cherokee [1] uses far less code than Apache. The Cherokee web server made waves when released early in 2006 (Figure 1), but Cherokee has not been widely adopted so far. Its benefit is that it can cover most critical web server applications without too much bloat. This economy becomes evident at the installation phase. Besides the C library, you do not need any special libraries, although Cherokee does depend on OpenSSL or GNUTLS for SSL connections. If you do without some functionality, you can even build a micro-variant of Cherokee in embedded systems.

After downloading [1] and unpacking, just follow the normal steps of *configure*, *make*, and *make install*. To install the Cherokee files in the standard directory hierarchy, you need to set the *--prefix = /usr* option in the configure phase. If you do not, the files will be dropped into */usr/local*. The configuration and webroot directories are set by the *--sysconfdir* and *--with-wwwroot* parameters, respectively.

For more security, Cherokee can also run in a chroot jail, and it has PAM authentication and LDAP interfaces. To improve Cherokee's performance in handling larger files, it can use the new

sendfile() Linux kernel call to accelerate data transfer between file descriptors and sockets. Cherokee relies on the kernel's Epoll interface for standard connection handling.

Sites as Symlinks

After setting the configuration directory to */etc* in the configure phase, the files with the Cherokee settings will reside below */etc/cherokee* – *cherokee.conf*, for example. If you have ever tinkered with Apache configuration, you should have no difficulty finding your way around, even if the syntax looks slightly different. Cherokee will also run if you just leave the defaults.

In *cherokee.conf*, the main settings reside that apply to all virtual servers: Servers are defined in *etc/cherokee/sites-enabled*. To be more precise, this is where the symbolic links are stored: The links point to files below *sites-available*, which is located at the same level. Administrators can use the symlinks to enable and disable sites without touching the files.

In a similar approach, the directories *mods-available* and *mods-enabled* contain the extension modules: SSL and server management, for example. Listing 1 shows an excerpt from the default server configuration.

Handlers, which enable specific Cherokee file-handling modules on the basis of file extensions or directory names, are particularly interesting. For example, the *file* handler tells the server to store the files it serves up internally in cache memory. Administrators can also enable PHP support in the same way, as shown in Listing 1, but this approach only gives an extremely slow CGI version of PHP.

Listing 2 shows how to enable the faster FastCGI (FCGI) interface. Typing *php-cgi -v* tells you whether the PHP CGI interpreter can actually handle FastCGI. FCGI or the SCGI interface, which is also available, allows administrators to integrate most other application servers or frameworks, such as Ruby on Rails.

El servidor Cherokee quiere ser una alternativa al dominio de Apache

Figure 1: Programmer Alvaro Lopez Ortega made a name for himself in the Spanish daily paper El País with the Cherokee Web server.

Lighty

Lighttpd [2], or Lighty, as the server is also known, plays in a completely different league. Just a couple of months ago it conquered a place in the top four of the Netcraft Web Server Charts [3], partly because a couple of major sites, such as *flickr.com*, use it – although not as their main server.

Programmer Jan Kneschke [4] has already made a name for himself with other open source projects, especially in the PHP field, and is still working on improving his server. Just like Cherokee, Lighty uses the Linux kernel's fast, state-of-art interfaces to accelerate request handling. For example, it will use the File Alteration Monitor (FAM) if needed, thus reducing the required number of *stat()* system calls. However, I did not use this feature in our lab.

Lighttpd not only impresses with excellent performance, but also with its versatile feature scope, which includes Flash streaming or Mod Magnet, which lets programmers control request handling phases via the Lua scripting language.

The configuration is convincingly simple, consisting of just one file. On the

Listing 1: Cherokee Default Configuration

```
01 DirectoryIndex index.php,
   index.html, index.htm, index.
   shtml
02
03 DocumentRoot /var/www/html
04
05 UserDir public_html {
06     Directory / {
07        Handler common
08     }
09
10     Directory /cgi-bin/ {
11          Handler cgi
12          DocumentRoot /var/
   www/cgi-bin/
13     }
14 }
15
16 Log combined {
17     AccessLog /var/log/
   cherokee.access
18     ErrorLog  /var/log/
   cherokee.error
19 }
20
21 Directory /icons {
22     Handler file
23     DocumentRoot /usr/share/
   cherokee/icons/
24 }
25
26 Extension php, php3, php4 {
27                Handler phpcgi
28 }
```

Listing 2: FastCGI

```
01 Extension php {
02    Handler fcgi {
03       Server localhost:8002 {
04          Env PHP_FCGI_MAX_
   REQUESTS "-1"
05          Env PHP_FCGI_CHILDREN
   "11"
06          Interpreter  "/usr/
   bin/php-cgi -b 8002"
07       }
08    }
09 }
```

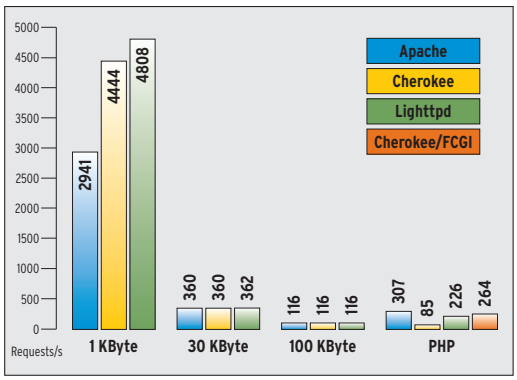

Figure 2: The benchmarks show that Cherokee and Lighttpd are faster than Apache with smaller files. The results are mixed for larger files and PHP scripts.

downside, the syntax is slightly complex and thus prone to error. Fortunately, web administrators can typically just comment out those parts of the sample file they do not need and replace the sample strings.

To enable FCGI, you need to load the corresponding server module (Listing 3). In theory, FCGI back-ends can be distributed over multiple servers: Lighttpd even has an integrated load balancer to help you do this.

Lighty will also run in a chroot environment if necessary, but it has far more functionality than Cherokee, including support for standards and features such as FCGI, SCGI, Proxy Function, Virtual Hosting, User Tracking, Traffic Shaping, and much more. The package also includes a couple of more exotic applications, such as Geo IP, which associates IP addresses with geographical locations, and a traffic visualization module for RRD Tool.

Hard Facts

In our lab, I asked Cherokee and Lighttpd to take on Apache in a small benchmark test. It has to be said that the benchmark, like any other web benchmark, can only give a rough indication of a server's expected behavior under production conditions. It is nearly impossible to simulate the complex behavior of a large number of clients with very different available bandwidths accessing a server at random times.

I did not enable any kind of optimization: all the servers used the standard configuration on an Athlon 2800+ with 1GB RAM. The benchmarks were measured on a dedicated 100Mbit Ethernet line with a single client. The client used Apachebench to access 1000 pages and process 10 requests in parallel. The command line for this was:

```
ab -c 10 -n 1000 ➥
http://Server/Data
```

My test data comprised three HTML files of 1KB, 30KB, and 100KB. As Figure 2 shows, the competitors were ahead of Apache with respect to smaller files. The benefits of lower processing overhead play an important role here. With larger files other factors are getting more important, and network bandwith is the limiting factor in the end.

For PHP scripts, Apache is well ahead, followed by Lighttpd, with Cherokee tailing behind. This ranking is not surprising because Cherokee uses the CGI variant of PHP by default, and this means spawning a new process for each access attempt. The situation improves drastically when Cherokee uses the FCGI interface to handle PHP. Doing so more than doubles Cherokee's performance, pushing it past Lighty, but without seriously endangering Apache's lead.

Light and Easy

Cherokee and Lighttpd are powerful web servers that outdo Apache in some respects. Of course, neither can hope to rival the functional scope that Apache offers. As the benchmarks show, the use of Cherokee and Lighttpd can boost performance if you simply need to serve up smaller, static files, as confirmed by the use of Lighttpd by the *flickr.com* site to serve up thumbnails on its server farm. In the opinion of this test team, both servers are easier to configure than Apache, a fact that made for a fairly uncomplicated lab session. ■

INFO

[1] Cherokee: *http://www.cherokee-project.com*

[2] Lighttpd: *http://www.lighttpd.net*

[3] Netcraft survey: *http://news.netcraft.com/archives/web_server_survey.html*

[4] Jan Kneschke's blog: *http://blog.lighttpd.net*

Listing 3: lighttpd.conf

```
01 server.modules = (  modules
02 #                 "mod_rewrite",
03 #                 "mod_redirect",
04 #                 "mod_alias",
05                   "mod_access",
06 #                 "mod_cml",
07 #                 "mod_trigger_b4_dl",
08 #                 "mod_auth",
09 #                 "mod_status",
10 #                 "mod_setenv",
11                   "mod_fastcgi",
12 ...
13 server.document-root  = "/var/www/html/"
14 server.errorlog       = "/var/log/lighttpd/error_
   log"
15 index-file.names      = ( "index.php", "index.
   html",
16                           "index.htm", "default.
   htm" )
17 ...
18 fastcgi.server        = ( ".php" =>
19                           ( "localhost" =>
20                             (
21                               "socket" => "/tmp/
   php-fastcgi.socket" ,
22                               "bin-path" => "/
   usr/bin/php-cgi"
23                             )
24                           )
25                         )
```