Boosting Apache with Perl

by
Jonathon Coombes

CALU 1999

Introduction

Apache Web Server

- Apache Configuration
- Apache Tuning
- Apache API

Mod_Perl

- Mod_Perl configuration
- Mod_Perl optimisation
- Converting CGI to Mod_Perl
- Writing Handlers

Embedded Perl Packages

- ePerl
- Mason
- Embedded Perl (embPerl)

Apache History

- In February, 1995, the most popular WWW server was the public domain NCSA HTTP server.
- Many webmasters developed their own extensions and bug fixes.
- A number of webmasters formed a mailing list, and shared the information between core developers.
- At the end of February, eight core contributors formed the original Apache Group foundation.
- Using NCSA httpd 1.3 as a base, the first official public release (0.6.2) of the Apache server was made in April 1995.
- Apache 0.8.8 was later released in August with benefits such as:
  - ✔ a modular structure and API
  - ✔ better extensibility
  - ✔ an adaptive pre-forking process model
  - ✔ pool-based memory allocation
- Apache 1.0 was released on Dec 1, 1995 after beta testing, porting to different platforms, and a new set of documentation were added to a standard set of modules.
- Less than a year later, the Apache server became the most used server on the Internet.

Apache Installation

Installation may be simple using a Red Hat package or similar binary package:

```
# rpm -ivh apache-1.3.6-1.i386.rpm
```

Or, it may be more complicated, by compiling the source:

```
# gzip -cd apache-1.3.6-1.tar.gz  |  tar xvf -
# cd apache-1.3.6

# ./configure --prefix=/usr/local/apache
# make
# make install
```

To start the Apache server, it is simply a matter of typing:

```
# /usr/local/apache/sbin/apachectl start
```

or

```
# /etc/rc.d/init.d/apache start
```

Dynamic Shared Objects (DSO)

- Dynamic shared objects allows a program to be built so it can be loaded at run-time into the actual executable address space.

- The loading is done in one of two ways:

  Automatically - using the system ld.so library

  Manually - using an API which gives dlopen()/dlsym() system calls.

- The first method produces shared libraries and are linked at build time using the -lfoo linker option.
- The second method produces shared objects which have to be manually by the executable into its address space.

- Symbols are resolved for the manual method by using an exported set of symbols from the executable and loaded DSO modules.

- Shared libraries are used commonly on most operating systems.
- The shared objects approach is only used by a small number of main stream programs such as Perl 5, Netscape Server and now Apache (as of version 1.3).

Dynamic Shared Objects (DSO)

Advantages

✔ Server is more flexible. It can be run as SSL, mod_perl or php with only one installation.

✔ Servers can be extended with other modules even after installation.

✔ Easier module development and testing as the compiling apache source is not required each time the module is changed.

Disadvantages

✗ DSO is not supported on all platforms.

✗ Startup of the server is slower due to symbol resolving.

✗ Can produce a slightly slower server depending on platform and address resolutioning.

✗ DSO modules cannot be linked with other DSO modules. This requires either the code be referenced directly through the Apache core, or compile Apache with chaining available.

✗ Some platforms cannot force the linker to export all global symbols for linking DSO and Apache executables. This is overcome using the SHARED_CORE feature of Apache and is used by default on such platforms.

Compiling DSO Modules

Build and install a distributed Apache module, say mod_foo.c, into its own DSO mod_foo.so:

Build and install using configure:

```
$ ./configure --prefix=/usr/local/apache
                --enable-shared=foo
$ make install
```

Build and install manually:

- Edit src/Configuration:
  - << AddModule modules/xxxx/mod_foo.o
  - >> SharedModule modules/xxxx/mod_foo.so

```
$ make
$ cp src/xxxx/mod_foo.so /usr/local/apache/libexec
```

- Edit /usr/local/apache/etc/httpd.conf
  - >> LoadModule foo_module /usr/local/apache/libexec/mod_foo.so

Compiling DSO Modules (continued...)

To build and install a third-party Apache module, say mod_foo.c, into the DSO mod_foo.so

Build and install using configure:

```
$ ./configure --add-module=/usr/local/src/mod_foo.c
                    --enable-shared=foo
$ make install
```

It can be built manually the same way as previously mentioned, but the source must first be copied into the appropriate Apache directory. This is the $APACHE_HOME/src/modules/extra directory.

To build and install a third-party Apache module, say mod_bar.c, into its own DSO mod_bar.so residing outside of the Apache source tree:

Build and install using the apxs package:

```
$ cd /usr/local/src/bar_src
$ apxs -c mod_bar.c
$ apxs -i -a -n bar mod_bar.so
```

Apache Fine Tuning

mod_status and ExtendedStatus On

Set ExtendedStatus off (the default) to reduce the number of calls to time functions.

DYNAMIC_MODULE_LIMIT=0

Saves RAM that would normally be allocated for supporting dynamically loaded modules.

HostnameLookups

Better performance is available by using IP Addresses rather than hostnames.

If DNS names are required in some CGI's, doing the gethostbyname() call in the script would be efficient and suitable.

FollowSymLinks and SymLinksIfOwnerMatch

Extra system calls are required to check symlinks with these two options.

AllowOverride

The possible .htaccess files has to be opened each time a directory is accessed.

Apache Fine Tuning (Continued...)

Negotiation

Use a sorted list of index files to take advantage of content negotiation:

```
DirectoryIndex index.cgi index.pl index.shtml index.html
```

Process Creation

- The MinSpareServers, MaxSpareServers, and StartServers options no longer dramatically effect on server performance.
- Set the MaxRequestsPerChild to zero for Linux and most platforms.
- Set MaxRequestsPerChild to 10000 for any memory leaks.
- MaxRequestsPerChild should not be set to a low number where performance is required.

Perl History

- First developed by Larry Wall who wanted an easier scripting language than that of the shell and awk packages.

Perl 0	Introduced Perl to my officemates.
Perl 1	Released Perl to the world, and changed /\(...|...\)/ to /(...|...)/.
Perl 2	Henry Spencer's regular expression package was added.
Perl 3	Added the ability to handle binary data.
Perl 4	Introduced the first Camel book.
Perl 5	Introduced a lot of new features.

CGI Scripts

- Common Gateway Interface (CGI) scripts are the most common form of computer/web interaction.
- CGI scripts can be written in a number of languages. The most common are the Perl and C/C++ languages.
- The language used for scripts is often based on the server architecture and/or the preference of the programmer.
- The C/C++ languages are compiled to produce an executable script, whereas the Perl executes and interpreter each time a Perl CGI script is accessed.
- Efficiency could be inproved by having only a single Perl interpreter running in memory, and passing it the Perl scripts. This is where mod_perl comes in to the picture.

Mod_Perl

Mod_Perl is an integration project which brings together the full power of the Perl programming language and the Apache HTTP server.

It provides a single embedded Perl interpreter within the Apache web server. This can be either statically, or as a DSO module.

Some of the advantages of mod_perl include:

- Able to write Apache modules entirely in Perl.
- Having a persistent interpreter in the server saves on overheads due to starting a perl interpreter for each script.
- Offers code caching, where the modules and scripts are being loaded and compiled only once.
- Increased power and speed.
- Full access to the web server.
- Allows customized processing of URI to filename translation, authentication, response generation and logging.
- Practically no run-time overhead.
- Improved performance of %200 - %2000 apparently obtained.

Mod_Perl Installation

Installation of a pre-built package

```
# rpm -ivh mod_perl-1.19-1.i386.rpm
```

Build and install manually

Start with the source for both Apache and mod_perl at the same directory level.

```
$ perl Makefile.PL APACHE_SRC=../apache_1.3.6/src \
  DO_HTTPD=1  USE_APACI=1 EVERYTHING=1 \
  PERL_MARK_WHERE=1
$ make
$ make test
$ make install
```

The following option can be added to the perl line if you are not root, or require a second server.

```
APACHE_PREFIX=/www/server PREFIX=/www/server
```

Tuning Mod_Perl

Reducing Memory Usage

- Set a maximum on the number of httpd processes to just larger than the RAM available.
- Pre-load commonly used modules so that the code is shared by all processes.
- Apache::Registry programs can be pre-loaded using the Apache::RegistryLoader module. This allows the code for such programs to be shared by all httpd processes.

Reducing Large Processes

- Put static content on one machine, mod_perl programs on another.
- Two servers run on one machine, each bound to a particular IP address.
- Use two port numbers rather than ip addresses. This may cause problems with firewalls.
- Use the ProxyPass option to refer all mod_perl pages to a different server or IP address.
- Use an accelerated proxy such as Squid.

Mod_Perl Action

Mod_Perl offers two ways to interface with Apache:

1. Perl interpreter embedded in Apache. This allows CGI scripts to be executed directly through the Apache server.
2. Full access to the Apache API.

Mod_Perl also allows the advantage of backward compatibility with the CGI interface standard.

Two modules provide this capability:

Apache::Registry	Used to run existing perl CGI scripts transparently.
Apache::PerlRun	Similar to Registry, but allows scripts which are not laid out suitably.

The httpd server and handlers can be configured in Perl using the PerlSetVar, and <Perl> sections.

Configuration directives can also be defined.

Perl Web Modules

Some of the more common PerlHandler modules:

ASP	Implement Active Server Pages
BBS	BBS like System for Apache
Embperl	Embed Perl in HTML
EmbperlChain	Feed handler output to Embperl
ePerl	Fast emulated Embedded Perl (ePerl)
FTP	Full-fledged FTP proxy
Gateway	A multiplexing gateway
GzipChain	Compress files on the fly
Mason	Build sites w/ modular Perl/HTML blocks
NavBar	Navigation bar generator
OutputChain	Chain output of stacked handlers
PerlRun	Run unaltered CGI scripts
Registry	Run unaltered CGI scripts
Session	Maintain session state across HTTP requests
SSI	Implement server-side includes in Perl
SSIChain	SSI on other modules output

A popular Server Configuration module:

PerlSections	Utilities for <Perl> sections

Configuring Apache for Mod_Perl

The Apache::Registry can be used to run all CGI scripts currently used in a web server.

The Apache::Registry module will allow CGI scripts to load much faster using the embedded mod_perl interpreter.

Configuration is done by changing the cgi-bin directory options with the httpd.conf file. Add the following information:

```
<Location /cgi-bin>
  SetHandler  perl-script
  PerlHandler Apache::Registry
  PerlSendHeader On
  Options +ExecCGI
</Location>
```

NOTE: This may have to be added to the srm.conf file, depending on the version and setup of the Apache server.

Apache API

The Apache Application Program Interface (API) allows the programmer to interact with the Apache server, by way of code.

The Apache API allows for much better control and improved performance compared to simply running the CGI script through the Registry module.

Mod_Perl can use the Apache API to control every phase of a connection.

These stages are:

1. URI->Filename translation
2. Header parsing
3. Access control
4. Authentication
5. Authorization
6. MIME type checking
7. Response
8. Logging

The Apache API can also be used by mod_perl to configure the server from within Perl.

A Simple Handler

Add the following configuration to the httpd.conf (or srm.conf) file.

```
<Location /helloworld>
   SetHandler  perl-script
   PerlHandler Apache::HelloWorld
</Location>
```

The perl code for the HelloWorld handler is:

```
package Apache::HelloWorld;

use strict vars;
use Apache::Constants ':common';

sub handler {
     my $r = shift;
     $r->content_type('text/html');
     $r->send_http_header;
     $r->print(<<END);
  <HTML>
  <HEADER>
  <TITLE>Hello World</TITLE>
  </HEADER>
  <BODY>
  <H1>Hello World!</H1>
  </BODY>
  </HTML>
  END
  return OK;
}
1;
```

Perl Extensions

ePerl

ePerl provides a form of embedded perl. This can be done using the Apache::ePerl module which can emulate ePerl in a very fast way.

Embperl

Embperl gives you the power to embed Perl code in your HTML documents. You can also use hundreds of Perl modules which have already been written - including DBI - for database access to a growing number of database systems.

Embperl has several features especially for HTML including dynamic tables, formfield-processing, and escaping/unescaping.

Mason

Mason is a Perl-based web site development and delivery engine. Mason provides facilities for both embedded HTML and many common web development issues such as:

templating, caching, debugging, profiling, page previewing

EmbPerl

The different embedded perl options are shown below:

[+ Perl code +]

Replace the command with the result you get from evaluating the Perl code. The Perl code can be anything which can be used as an argument to a Perl eval statement.

[+ $a +] Replaced with the content of the variable $a

[- Perl code -]

Executes the Perl code, but deletes the whole command from the HTML output.

[- $a=1 -] Set the variable $a to one.

[- $i=0; while ($i<5) {$i++} -]

[! Perl Code !]

Same as [- Perl Code -] with the exception that the code is only executed at the first request. This could be used to define subroutines, or do one-time initialization.

[# Comments #]

This is a comment block. Everything between the [# and the #] will be removed from the output.

EmbPerl (continued...)

[* Perl code *]

This is similar to [- Perl Code -], the main difference is, while [- Perl Code -], has always it's own scope, all [* Perl code *] blocks runs in the same scope. If you like to use perl's control structures, Perl's if, while, for etc. can not spawn mulitple [- Perl Code -] blocks, but can spawn multiple [* Perl Code *] blocks.

```
[* foreach $i (1..10) { *]
  [- $a = $i + 5 -]
  loop count + 5 = [+ $a +] <br>
[* } *]
```

The following will not work:

```
[- foreach $i (1..10) { -]
  some embedded text or commands
[- } -]
```

This is normally done using the Embperl metacommands as follows:

```
[$ foreach $i (1..10) $]
[- $a = $i + 5 -]
  loop count + 5 = [+ $a +] <br>
[$ endforeach $]
```

Embperl Meta-Commands

[$ Command Args $]

Execute an Embperl metacommand. Command can be any one of the following.

if, elsif, else, endif

Standard conditional statements

eg [$ if $age > 12 and $age < 20 $]
Hi, teenager!
[$ endif $]

while, endwhile

do, until

Standard conditional loop statements

eg [$ while $i < 10 $]
The current loop value is [+ $i +]
[$ endwhile]

foreach, endforeach

Recurse through an element list

eg [$ foreach $num (2,4,6) $]
The current value is [+ $num +]
[$ endforeach $]

hidden - Declare hidden fields in a form

var - Declare strict variables

Embperl HTML Tags

Embperl recognizes the following HTML tags specially. All others are simply passed through, as long as they are not part of a Embperl command.

TABLE, /TABLE, TR, /TR

Embperl can dynamically generate tables. Embperl uses the special variables $row, $col, and $cnt in constructing tables.

TH, /TH

Embperl defines a table heading.

DIR, MENU, OL, UL, DL, SELECT, /DIR, /MENU, /OL, /UL, /DL, /SELECT

Lists, menus and form selections.

OPTION

INPUT

All values of <INPUT> tags are stored in the hash %idat, with NAME as the hash key and VALUE as the hash value.

TEXTAREA, /TEXTAREA

The TEXTAREA tag is treated exactly like other input fields.

META HTTP-EQUIV= ...

<meta http-equiv= ... > will over-ride the corresponding http header.

'Hello World' Example

Embedded Perl would not require any special code for a simple "Hello World" example. Instead, the example uses a simple foreach loop to change the heading level of the string.

```
<HTML>
<HEAD>
<TITLE>Hello World</TITLE>
</HEAD>
<BODY>
  [- @arr = (1, 2, 3, 4, 5) -]
  [$ foreach $level @arr $]
  <H[+ $level +]>Hello World</H[+ $level +]><BR>
  [$ endforeach $]
</BODY>
</HTML>
```

Produces output similar to the following:

<h1>Hello World</h1>
<h2>Hello World</h2>
<h3>Hello World</h3>
<h4>Hello World</h4>
<h5>Hello World</h5>

Dynamic Table Example

```
[- @env = keys %ENV -]
    Row              Var                    Content
[+ $i=$row +] [+ $env[$row] +]   [+ $ENV{$env[$i]} +]
```

Produces output similar to the following:

Row	Var	Content
0	SERVER_SOFTWARE	Apache/1.3.4 (Unix)
1	DOCUMENT_ROOT	/www/htdocs
2	GATEWAY_INTERFACE	CGI-Perl/1.1
3	REMOTE_ADDR	123.213.231.132
4	SERVER_PROTOCOL	HTTP/1.1
5	REQUEST_METHOD	GET

...

More specific tables can be created manually

```
[-
$table[0][0] = ’1/1’ ;
$table[1][0] = ’2/1’ ;
$table[1][1] = ’2/2’ ;
$table[2][0] = ’3/1’ ;
$table[2][1] = ’3/2’ ;
$table[2][2] = ’3/3’ ;
$maxcol=5 ;
-]
[+ $a[$row][$col] +]
```

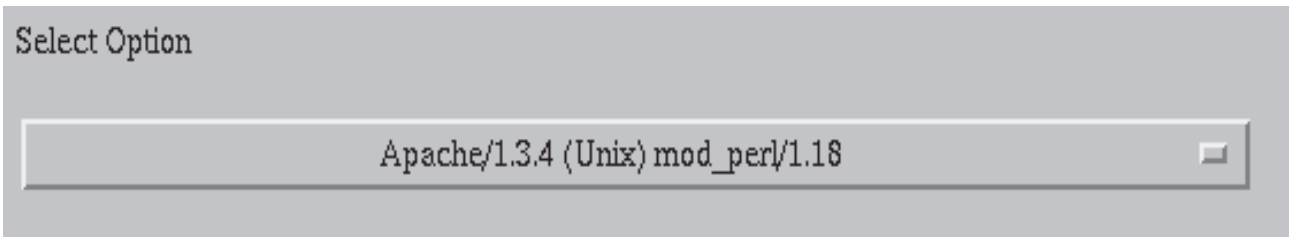
1/1		
2/1	2/2	
3/1	3/2	3/3

Simple Form Example

```
[- @k = keys %ENV -]
[- @v = values %ENV -]

<form method="POST">
<P>Select Option</P>
<p>
<select name="opt">
<option value="[+ $k[$row] +]">[+ $v[$row] +]</option>
</select>
</p>
</form>
```

This form would present



with a pull list of all the environment settings currently on the system.

Database Example

```
[-
        $sth = $dbh -> prepare (
        "SELECT id,cost from $table \
         where cost > 100");

         $sth -> execute ;
-]

Product ID     Cost
[$ while $dat = $sth -> fetchrow_arrayref $]
[+ @$dat[0] +]      [+ @$dat[1] +]<BR>
[$ endwhile $]
```

This would produce output like:

Product ID	Cost
201	$120
250	$195
277	$155
385	$325
401	$223

Complex Example

Combining the form and database examples allows for a more complex, but appropriate presentation on a web site.

```
<FORM ACTION="/cgi-bin/purchase"  NAME="CDROM" METHOD=POST>
[-
   $sth = $dbh -> prepare (
        "SELECT id,description,cost  \
         from $table \
         where band like 'AC/DC'
         order by release_date") ;

   $sth -> execute ;
-]
<TABLE>
<TR><TH COLSPAN=3>Music CDROM's</TH><TR>
<TR><TH>ID</TH><TH>Description</TH><TH>Cost</TH></TR>
[$ while $dat = $sth -> fetchrow_arrayref $]
<TR>
<TD><INPUT TYPE=TEXT SIZE=3 NAME=[+ @$dat[0] +]></TD>
<TD WIDTH=80%><B>[+ @$dat[1] +]</B></TD>
<TD>$[+ @$dat[2] +]</TD>
</TR>
[$ endwhile $]
</TABLE>

<INPUT TYPE=HIDDEN NAME="CDROM" VALUE="Music">
<INPUT TYPE=SUBMIT VALUE="Purchase CDROM's">
</FORM>
```

Summary

- Apache is still considered one of the most powerful and flexible web servers available today.
- Mod_Perl allows an improved efficiency and finer control over the interaction with the Apache API.
- Tuning both Apache and Mod_Perl to suit the server's situation will improve performance.
- Mod_Perl allows a Perl script to be executed using an embedded interpreter, rather than executing a new interpreter for each run.
- Writing a handler gives better performance under Apache, but can be more complicated than a simple script.
- Old-style CGI scripts can still be executed under mod_perl using the Apache::Registry, or Apache::PerlRun modules.
- Embedded perl is available in a number of different packages including Embperl, ePerl, and Mason.
- Embedded perl offers a simple, yet powerful method of connection between the user and the Apache API or system.

Further Information

Information on the Apache web server:

- http://www.apache.org
- http://www.refcards.com/
- Apache: The Definitive Guide, Ben Laurie and Peter Laurie; O'Reilly & Associates (1999).
- Apache Server for Dummies, by Ken Coar; IDE (1998).

Information on mod_perl:

- http://perl.apache.org
- http://www.modperl.com
- http://forum.swarthmore.edu/epigone/modperl
- http://www.refcards.com/
- Writing Apache Modules with Perl and C, Lincoln Stein and Doug MacEachern; O'Reilly & Associates (1999).

Information on embedded perl:

- http://perl.apache.org/embperl/
- http://www.masonhq.com
- http://www.engelschall.com/sw/eperl/