1 mod_perl for ISPs. mod_perl and Virtual Hosts

1.1 Description

mod_perl hosting by ISPs: fantasy or reality? This section covers some topics that might be of interest to users looking for ISPs to host their mod_perl-based website, and ISPs looking for a way to provide such services.

Today, it is a reality: there are a number of ISPs hosting mod_perl, although the number of these is not as big as we would have liked it to be. To see a list of ISPs that can provide mod_perl hosting, see ISPs supporting mod_perl.

Note: At this moment this document talks about mod_perl 1.0. mod_perl 2.0 coupled with the perchild mpm (http://httpd.apache.org/docs-2.0/mod/perchild.html) will allow different users run mod_perl handlers under different uid/gid. This solves the problem of secure co-existing of more than one mod_perl user on the same httpd server.

1.2 ISPs providing mod_perl services - a fantasy or a reality

- You installed mod_perl on your box at home, and you fell in love with it. So now you want to convert your CGI scripts (which currently are running on your favorite ISPs machine) to run under mod_perl. Then you discover that your ISP has never heard of mod_perl, or he refuses to install it for you.

- You are an old sailor in the ISP business, you have seen it all, you know how many ISPs are out there and you know that the sales margins are too low to keep you happy. You are looking for some new service almost no one else provides, to attract more clients to become your users and hopefully to have a bigger slice of the action than your competitors.

If you are a user asking for a mod_perl service or an ISP considering to provide this service, this section should make things clear for both of you.

An ISP has three choices:

1. ISPs probably cannot let users run scripts under mod_perl on the main server. There are many reasons for this:

   Scripts might leak memory, due to sloppy programming. There will not be enough memory to run as many servers as required, and clients will be not satisfied with the service because it will be slower.

   The question of file permissions is a very important issue: any user who is allowed to write and run a CGI script can at least read (if not write) any other files that belong to the same user and/or group the web server is running as. Note that it's impossible to run `suEXEC` and `cgiwrap` extensions under mod_perl 1.0.

   Another issue is the security of the database connections. If you use `Apache::DBI`, by hacking the `Apache::DBI` code you can pick a connection from the pool of cached connections even if it was opened by someone else and your scripts are running on the same web server.

Yet another security issue is a potential compromise of the systems via user's code running on the webservers. One of the possible solutions here is to use chroot(1) or jail(8) mechanisms which allow to run subsystems isolated from the main system. So if a subsystem gets compromised the whole system is still safe.

There are many more things to be aware of so at this time you have to say *No*.

Of course as an ISP you can run mod_perl internally, without allowing your users to map their scripts so that they will run under mod_perl. If as a part of your service you provide scripts such as guest books, counters etc. which are not available for user modification, you can still can have these scripts running very fast.

2. But, hey why can't I let my users run their own servers, so I can wash my hands of them and don't have to worry about how dirty and sloppy their code is (assuming that the users are running their servers under their own usernames, to prevent them from stealing code and data from each other).

   This option is fine as long as you are not concerned about your new systems resource requirements. If you have even very limited experience with mod_perl, you know that mod_perl enabled Apache servers while freeing up your CPU and allowing you to run scripts very much faster, have huge memory demands (5-20 times that of plain Apache).

   The size depends on the code length, the sloppiness of the programming, possible memory leaks the code might have and all that multiplied by the number of children each server spawns. A very simple example: a server, serving an average number of scripts, demanding 10Mb of memory which spawns 10 children, already raises your memory requirements by 100Mb (the real requirement is actually much smaller if your OS allows code sharing between processes and programmers exploit these features in their code). Now multiply the average required size by the number of server users you intend to have and you will get the total memory requirement.

   Since ISPs never say *No*, you'd better take the inverse approach - think of the largest memory size you can afford then divide it by one user's requirements as I have shown in this example, and you will know how many mod_perl users you can afford :)

   But you cannot tell how much memory your users may use? Their requirements from a single server can be very modest, but do you know how many servers they will run? After all, they have full control of *httpd.conf* - and it has to be this way, since this is essential for the user running mod_perl.

   All this rumbling about memory leads to a single question: is it possible to prevent users from using more than X memory? Or another variation of the question: assuming you have as much memory as you want, can you charge users for their average memory usage?

   If the answer to either of the above questions is *Yes*, you are all set and your clients will prize your name for letting them run mod_perl! There are tools to restrict resource usage (see for example the man pages for `ulimit(3)`, `getrlimit(2)`, `setrlimit(2)` and `sysconf(3)`, the last three have the corresponding Perl modules: `BSD::Resource` and `Apache::Resource`).

[ReaderMETA]: If you have experience with other resource limiting techniques please share it with us. Thank you!

If you have chosen this option, you have to provide your client with:

- Shutdown and startup scripts installed together with the rest of your daemon startup scripts (e.g */etc/rc.d* directory), so that when you reboot your machine the user's server will be correctly shutdown and will be back online the moment your system starts up. Also make sure to start each server under the username the server belongs to, or you are going to be in big trouble!

- Proxy services (in forward or httpd accelerator mode) for the user's virtual host. Since the user will have to run their server on an unprivileged port (>1024), you will have to forward all requests from `user.given.virtual.hostname:80` (which is `user.given.virtual.hostname` without the default port 80) to `your.machine.ip:port_assigned_to_user` . You will also have to tell the users to code their scripts so that any self referencing URLs are of the form `user.given.virtual.hostname`.

  Letting the user run a mod_perl server immediately adds a requirement for the user to be able to restart and configure their own server. Only root can bind to port 80, this is why your users have to use port numbers greater than 1024.

  Another solution would be to use a setuid startup script, but think twice before you go with it, since if users can modify the scripts they will get a root access. For more information refer to the section "SUID Start-up Scripts".

- Another problem you will have to solve is how to assign ports between users. Since users can pick any port above 1024 to run their server, you will have to lay down some rules here so that multiple servers do not conflict.

  A simple example will demonstrate the importance of this problem: I am a malicious user or I am just a rival of some fellow who runs his server on your ISP. All I need to do is to find out what port my rival's server is listening to (e.g. using `netstat(8)`) and configure my own server to listen on the same port. Although I am unable to bind to this port, imagine what will happen when you reboot your system and my startup script happens to be run before my rival's one! I get the port first, now all requests will be redirected to my server. I'll leave to your imagination what nasty things might happen then.

  Of course the ugly things will quickly be revealed, but not before the damage has been done.

  Luckily there are special tools that can ensure that users that aren't authorized to bind to certain ports (above 1024) won't be able to do so. One such a tool is called `cbs` and its documentation can be found at *http://www.epita.fr/~flav/cbs/doc/html*.

Basically you can preassign each user a port, without them having to worry about finding a free one, as well as enforce `MaxClients` and similar values by implementing the following scenario:

For each user have two configuration files, the main file, *httpd.conf* (non-writable by user) and the user's file, *username.httpd.conf* where they can specify their own configuration parameters and override the ones defined in *httpd.conf*. Here is what the main configuration file looks like:

```
httpd.conf
----------
# Global/default settings, the user may override some of these
...
...
# Included so that user can set his own configuration
Include username.httpd.conf

# User-specific settings which will override any potentially
# dangerous configuration directives in username.httpd.conf
...
...

username.httpd.conf
-------------------
# Settings that your user would like to add/override,
# like <Location> and PerlModule directives, etc.
```

Apache reads the global/default settings first. Then it reads the *Include*'d *username.httpd.conf* file with whatever settings the user has chosen, and finally it reads the user-specific settings that we don't want the user to override, such as the port number. Even if the user changes the port number in his *username.httpd.conf* file, Apache reads our settings last, so they take precedence. Note that you can use Perl sections to make the configuration much easier.

3. A much better, but costly solution is *co-location*. Let the user hook his (or your) stand-alone machine into your network, and forget about this user. Of course either the user or you will have to undertake all the system administration chores and it will cost your client more money.

   Who are the people who seek mod_perl support? They are people who run serious projects/businesses. Money is not usually an obstacle. They can afford a stand alone box, thus achieving their goal of autonomy whilst keeping their ISP happy.

1.2.1 Virtual Servers Technologies

As we have just seen one of the obstacles of using mod_perl in ISP environments, is the problem of isolating customers using the same machine from each other. A number of virtual servers (don't confuse with virtual hosts) technologies (both commercial and Open Source) exist today. Here are some of them:

- **The User-mode Linux Kernel**

  http://user-mode-linux.sourceforge.net/

  User-Mode Linux is a safe, secure way of running Linux versions and Linux processes. Run buggy software, experiment with new Linux kernels or distributions, and poke around in the internals of Linux, all without risking your main Linux setup.

User-Mode Linux gives you a virtual machine that may have more hardware and software virtual resources than your actual, physical computer. Disk storage for the virtual machine is entirely contained inside a single file on your physical machine. You can assign your virtual machine only the hardware access you want it to have. With properly limited access, nothing you do on the virtual machine can change or damage your real computer, or its software.

So if you want to completely protect one user from another and yourself from your users this might be yet another alternative to the solutions suggested at the beginning of this chapter.

- **VMWare Technology**

  Allows running a few instances of the same or different OSs on the same machine. This technology comes in two flavors:

  Open source: http://savannah.nongnu.org/projects/plex86/

  Commercial: http://www.vmware.com/

  So you may want to run a separate OS for each of your clients

- **freeVSD Technology**

  freeVSD (http://www.freevsd.org), an open source project sponsored by Idaya Ltd. The software enables ISPs to securely partition their physical servers into many *virtual servers*, each capable of running popular hosting applications such as Apache, Sendmail and MySQL.

- **S/390 IBM server**

  Quoting from: http://www.s390.ibm.com/linux/vif/

  "The S/390 Virtual Image Facility enables you to run tens to hundreds of Linux server images on a single S/390 server. It is ideally suited for those who want to move Linux and/or UNIX workloads deployed on multiple servers onto a single S/390 server, while maintaining the same number of distinct server images. This provides centralized management and operation of the multiple image environment, reducing complexity, easing administration and lowering costs."

  In two words, this a great solution to huge ISPs, as it allows you to run hundreds of mod_perl servers while having only one box to maintain. The drawback is the price :)

  Check out this scalable mailing list thread for more details from those who know: http://archive.develooper.com/scalable@arctic.org/msg00235.html

1.3 Virtual Hosts in the guide

If you are about to use *Virtual Hosts* you might want to read these sections:

Apache Configuration in Perl

Easing the Chores of Configuring Virtual Hosts with mod_macro

Is There a Way to Provide a Different startup.pl File for Each Individual Virtual Host

Is There a Way to Modify @INC on a Per-Virtual-Host or Per-Location Basis.

A Script From One Virtual Host Calls a Script with the Same Path From the Other Virtual Host

1.4 Maintainers

Maintainer is the person(s) you should contact with updates, corrections and patches.

- Stas Bekman <stas (at) stason.org>

1.5 Authors

- Stas Bekman <stas (at) stason.org>

Only the major authors are listed above. For contributors see the Changes file.

Table of Contents: