

# Performance Tuning for `mod_perl` Enabled Web Sites

Vivek Khera, Ph.D.  
Khera Communications, Inc.

Perl Conference 2.0  
August 20, 1998

---

## **Abstract**

---

I will be providing examples and hints on how to configure `mod_perl` enabled Apache web servers for high-speed performance. The strategy behind these techniques is to reduce the resource requirements of the `mod_perl` enabled `httpd` processes.

---

## Assumptions

---

It is assumed that you are using `mod_perl` for the speed benefits to replace your traditional CGI programs.

The speed benefit comes from:

- Not forking a separate process.
- Keeping persistent state such as database handles.
- Not recompiling the perl code every time it is used.

---

## Why do I need to do anything special?

---

- httpd processes are significantly larger with `mod_perl` than without
- On a very busy site, the *number* of httpd processes can grow to be quite large.

---

## **The Problem**

---

Response time slow due to thrashing or excessive paging.

---

## **Solution**

---

Reduce resource requirements:

1. Reduce memory footprint of each httpd process.
2. Reduce the number of large httpd processes.

---

## **Reducing memory footprint of httpd process**

---

Three approaches to reducing memory consumption by each httpd process:

1. Share memory for common Perl modules.
2. Share memory for commonly used programs.
3. Avoid memory leaks in programs.

---

## Sharing code for modules and programs

---

- To make code shared by all httpd processes, it must be loaded into the parent httpd process before it spawns children.
- Use the `PerlRequire` or `PerlModule` configuration directives to load the code.

---

## Sample PerlRequire script

---

Example PerlRequire:

```
#!/usr/local/bin/perl
use strict;

# Common modules
use Apache::Registry ();
use CGI (); CGI->compile(':all');
use CGI::Carp ();
use DBI ();
use DBD::mysql ();

# Perl program code used
use Apache::RegistryLoader ();

my $r = Apache::RegistryLoader->new;
$r->handler('/programs/play');
$r->handler('/programs/bonus');
$r->handler('/programs/claim');

1;
```

---

## Problem of memory leakage

---

Memory leaks in programs affect `mod_perl` more than traditional programs:

- Memory allocated (scalars, arrays, etc.) by `mod_perl` programs are not released to the OS until the process terminates.
- Because `mod_perl` exists for the life of the `httpd` process, memory allocated will not be released to the OS until the `httpd` child process terminates.
- Memory allocated and subsequently released by Perl will not be reclaimed by the OS, but it will be re-used by the `mod_perl` process for subsequent Perl programs it executes.

---

## Reducing risk of memory leaks

---

- Avoid global variables.
- Always use the `use strict; pragma.`
- Always turn on Perl warnings (`PerlWarn On`).
- For good measure, turn on taint checks, too (`PerlTaintCheck On`).

---

## **Memory use is still high**

---

Even after reducing the size of each httpd, the overall memory usage on the system can still be too high.

Options:

- Add more RAM.
- Reduce number of memory-consuming processes.

---

## Reducing number of large httpd processes

---

Run two httpd configurations — one for `mod_perl` dynamic content and one for static content.

Options:

- Two machines
- Two IP addresses on same machine
- Two ports on same machine

---

## Running httpd on two machines

---

- Simplest method.
- Must ensure links point to the proper httpd.
- Requires additional hardware.

---

## Example

---

Site developed to promote launch of new web browser for large west-coast company:

1. Large Windows NT system running IIS handled all static content, and one dynamic page which tested for the version of browser software. If accepted, displayed link to backend server.
2. Average Solaris box running `mod_perl` handled requests by accessing an SQL database and a Berkeley DB database.

---

## Running httpd on two IP addresses

---

- Similar to two machines method.
- Both httpd configurations run on same host.
- Requires additional IP address.

---

## Running httpd on two ports

---

- Instead of binding httpd to different IP addresses, bind to different port numbers.
- Firewalls might prevent access to the alternate port.

---

## Hiding split servers from users

---

A proxy server can be used to hide the dual server configurations from users, or to deliver the static content directly.

Two configurations are discussed:

- Squid in accelerator mode.
- Apache with `ProxyPass` directive.

---

## Example

---

Promotion to increase subscribership to a large online service:

- PentiumPro 200 with 256Mb RAM running BSD/OS.
- Httpd configured as normal, but uses port 9066.
- Squid accelerator on port 80 handled over 10 million hits per week for graphics and html files. Cached dynamic files which changed rarely.
- Httpd handled only about 50,000 requests per day. Each request resulted in two to three SQL queries.
- Never required more than 20 httpd children. Never used swap space.

---

## Configuration of ProxyPass

---

```
# forward to perl-enabled httpd
ProxyPass /programs
    http://localhost:8042/programs
```

### Assumptions:

- Static content httpd is on port 80.
- `mod_perl` httpd is on port 8042 on same machine.
- Apache is configured with Proxy module.

Benefit: you don't need to tweak any URLs to make sure they point to the right server.

---

## Example

---

Promotion to increase viewership of TV channel:

- Pentium 200 running BSD/OS, 128Mb RAM.
- Used ProxyPass to hide expensive `mod_perl` server from the world.
- Was able to handle tens of thousands of expensive queries to a database per day.

---

## What about all dynamic content?

---

So far, we have assumed that the majority of the content on a site is static, or otherwise not processed by `mod_perl`.

This is not always the case.

---

## All dynamic content

---

If all of your requests require processing by `mod_perl`, then your options are:

1. Throw a *lot* of memory on your machine.
2. Try to tweak the perl code to be as small and lean as possible.
3. Share the virtual memory pages by pre-loading the code.
4. Spread load across multiple identically configured servers.
5. Put all graphics on separate server.

---

## Summary

---

To gain maximal performance of `mod_perl`, one must reduce the resources used by the `httpd` to fit within what the machine has available.

Using the techniques described here can help provide much more performance and capacity from existing hardware.