1   Performance Considerations Under Different MPMs

1.1 Description

This chapter discusses how to choose the right MPM to use (on platforms that have such a choice), and how to get the best performance out of it.

Certain kind of applications may show a better performance when running under one mpm, but not the other. Results also may vary from platform to platform.

CPAN module developers have to strive making their modules function correctly regardless the mpm they are being deployed under. However they may choose to indentify what MPM the code is running under and do better decisions better on this information, as long as it doesn't break the functionality for other platforms. For examples if a developer provides thread-unsafe code, the module will work correctly under the prefork mpm, but may malfunction under threaded mpms.

1.2 Memory Requirements

Since the very beginning mod_perl users have enjoyed the tremendous speed boost mod_perl was providing, but there is no free lunch -- mod_perl has quite big memory requirements, since it has to store the compiled code in the memory to avoid the code loading and recompilation overhead for each request.

1.2.1 Memory Requirements in Prefork MPM

For those familiar with mod_perl 1.0, mod_perl 2.0 has not much new to offer. We still rely on shared memory, try to preload as many things as possible at the server startup and limit the amount of used memory using specially designed for that purpose tools.

The new thing is that the core API has been spread across multiply modules, which can be loaded only when needed (this of course works only when mod_perl is builts as DSO). This allows to save some memory. However the savings are not big, since all these modules are writen in C, making them into the text segments of the memory, which is perfectly shared. The savings are more significant at the startup speed, since the startup time, when DSO modules are loaded, is growing almost quadratically as the number of loaded DSO modules grows (because of symbol relocations).

1.2.2 Memory Requirements in Threaded MPM

The threaded MPM is a totally new beast for mod_perl users. If you run several processes, the same memory sharing techniques apply, but usually you want to run as few processes as possible and to have as many threads as possible. Remember that mod_perl 2.0 allows you to have just a few Perl interpreters in the process which otherwise runs multiple threads. So using more threads doesn't mean using significantly more memory, if the maximum number of available Perl interpreters is limited.

Even though memory sharing is not applicable inside the same process, mod_perl gets a significant memory saving, because Perl interpreters have a shared opcode tree. Similar to the preforked model, all the code that was loaded at the server startup, before Perl interpreters are cloned, will be shared. But there is a significant difference between the two. In the prefork case, the normal memory sharing applies: if a single byte of the memory page gets unshared, the whole page is unshared, meaning that with time less

and less memory is shared. In the threaded mpm case, the opcode tree is shared and this doesn't change as the code runs.

Moreover, since Perl Interpreter pools are used, and the FIFO model is used, if the pool contains three Perl interpreters, but only one is used at any given time, only that interpreter will be ever used, making the other two interpreters consuming very little memory. So if with prefork MPM, you'd think twice before loading mod_perl if all you need is trans handler, with threaded mpm you can do that without paying the price of the significantly increased memory demands. You can have 256 light Apache threads serving static requests, and let's say three Perl interpreters running quick trans handlers, or even heavy but infrequest dynamic requests, when needed.

It's not clear yet, how one will be able to control the amount of running Perl interepreters, based on the memory consumption, because it's not possible to get the memory usage information per thread. However we are thinking about running a garbage collection thread which will cleanup Perl interpreters and occasionaly kill off the unused ones to free up used memory.

1.3 Work with DataBases

1.3.1 Work with DataBases under Prefork MPM

`Apache::DBI` works as with mod_perl 1.0, to share database connections.

1.3.2 Work with DataBases under Threaded MPM

The current `Apache::DBI` should be usable under threaded mpm, though it doesn't share connections across threads. Each Perl interpreter has its own cache, just like in the prefork mpm.

`DBI::Pool` is a work in progress, which should bring the sharing of database connections across threads of the same process. Watch the mod_perl and dbi-dev lists for updates on this work. Once `DBI::Pool` is completed it'll either replace `Apache::DBI` or will be used by it.

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: