1 Discussion of multithreading on Win32 mod_perl 1.xx

1.1 Description

This document discusses the multithreading limitations of mod_perl-1.xx on Win32.

1.2 The problem

On Win32, mod_perl is effectively single threaded. What this means is that a single instance of the interpreter is created, and this is then protected by a server-wide lock that prevents more than one thread from using the interpreter at any one time. The fact that this will prevent parallel processing of requests, including static requests, can have serious implications for production servers that often must handle concurrent or long-running requests.

This situation changes with Apache/mod_perl 2.0, which is based on a multi-process/multi-thread approach using a native Win32 threads implementation. See the mod_perl 2 overview for more details, and the discussion of modperl-2 in Win32 on getting modperl-2 for Win32 in particular.

1.3 Does it really matter?

How serious is this? For some people and application classes it may be a non-problem, assuming the static material issue is handled differently.

Low traffic and single user development sites will likely be unaffected (though the lattest are likely to experience some surprises when moving to an environment where requests are no longer serialized and concurrency kicks in).

If your application is CPU bound, and all requests take roughly the same time to complete, then having more processing threads than processors (CPUs) will actually slow things down, because of the context switching overhead. Note that, even in this case, the current state of mod_perl will bar owners of multiprocessor Win32 machines from gaining any load balancing advantage from their superior hardware.

On the other hand, applications dealing with a large service times spread - say ranging from fractions of a second to a minute and above - stand to lose a great deal of responsiveness from being single threaded. The reason is that short requests that happen to be queueued after long ones will be delayed for the entire duration of the "jobs" that precede them in the queue; with multitasking they would get a chance to complete much earlier.

1.4 Workarounds

If you need multithreading on Win32, either because your application has long running requests, or because you can afford multiprocessor hardware, and assuming you cannot switch operating systems, you may want to consider a few workarounds and/or alternatives - which do not require waiting for 2.0.

You may be able to make Win32 multithreading a non-issue by tuning or rearranging your application and your architecture (useful tips on both counts can be found elsewhere in this document). You may be able to significantly reduce your worst-case timing problems or you may find that you can move the webserver

to a more mod_perl friendly operating system by using a multi-tier scheme.

If your application needs the full power of the Apache modules (often the case for people running outside Apache::Registry) you may want to consider a multi-server load balancing setup which uses mod_rewrite (or a similar URL partitioning scheme) to spread requests to several web servers, listening on different ports.

The mod_proxy dual server setup, discussed in the "Strategy" section, is also a possibility, although people who have tried it have reported problems with Win32 mod_proxy.

If you code to Apache::Registry (writing CGI compliant code) and can characterize the time demanded by a request from its URL, you can use a rewrite-based load balancing with a single server, by sending short requests to mod_perl while routing longer ones to the pure CGI environment - on the basis that startup, compilation and init times will matter less in this case.

If none of the above works for you, then you will have to turn to some non mod_perl alternatives: this, however, implies giving up on most of the flexibility of the Apache modules.

For CGI compliant scripts, two possible (portable) alternatives which are supported in an Apache/perl environment are straight CGI and FastCGI. In theory a CGI application that runs under mod_perl should have very few or no problems to run under straight CGI (though its performance may be unacceptable). A FastCGI port should also be relatively painless. However, as always, your mileage may vary.

If you do not mind replacing Apache with IIS/PWS, you may want to experiment with ActiveState’s value added PerlEx extension, which speeds up CGI scripts much in a way similar to what FastCGI does. PerlEx is transparently supported by CGI.pm, so users of this package should be more or less covered. (A IIS-FastCGI accelerator is, regrettably, no longer available.)

1.5 See Also

The mod_perl documentation and http://httpd.apache.org/, especially the discussion of Apache-2 and modperl-2. Help is also available through the archives of and subscribing to the mod_perl mailing list.

1.6 Maintainers

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

- Randy Kobes <randy@theoryx5.uwinnipeg.ca>

1.7 Authors

- Randy Kobes <randy@theoryx5.uwinnipeg.ca>

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

Table of Contents: