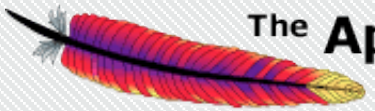
APACHE 2

Performance & Resource Tuning

OSCOM.4

Erik Abele



The Apache Software Foundation

http://www.apache.org/

Apache History

- 1994 NCSA HTTPd (University of Illinois)
- 1995 A "patchy server" is born
  - April: Apache 0.6.2 - first public release
  - December: Apache 1.0
- 1997 Apache 1.2.0
- 1998 Apache 1.3.0
- 1999 Incorporation of the ASF
- 2000 Apache 2.0 Alpha 1
- 2002 Apache 2.0 GA
- 2004 Apache 2.1 Beta 1 ?
- ...

Apache 2: New Features

- Based on the **A**pache **P**ortable **R**untime
- MPMs (Multi-processing modules)
- Filtering, IPv6 and Multi-protocol support
- Built-in SSL and improved Authn/Authz mechanisms (e.g. mod_auth_ldap)
- Module improvements
  - New: mod_dav, mod_deflate, mod_logio, ...
  - Improved: mod_include, mod_negotiation, ...
- Out-of-the-box XHTML-compliant, multi-language error responses
- Drastically improved module API
- Active development

Apache Portable Runtime

- Used by Apache HTTPD, Subversion, Flood, Prothon and other projects
- Consistent interface to underlying platform-specific implementations
- Platforms are implemented in their native APIs instead of using the POSIX-emulation layers
- Solid foundation for Linux, Unix and non-Unix platforms such as BeOS, OS/2 and Windows

What is performance?

- Performance = throughput, measured in successfully completed requests per second
- A statement of the speed at which the webserver works
- The degree to which the webserver fulfills the purposes for which it was built or acquired, or which it is now expected to fulfill; a function of effectiveness, reliability, and cost

➟ Performance tuning always means to find the adequate balance between a variety of needs: speed, features, flexibility, portability, stability, ...

Performance factors

- Performance depends on a broad and complex spectrum of different factors:
  - Hardware (RAM, CPU, HDD, ...)
  - Network resources (bandwith, latency, traffic)
  - Operating system
  - Compile-time and run-time configuration
  - Other services running on the same box
  - Structure of content to be served (e.g. static vs. dynamic pages, databases, proxied content, ...)
  - Real-world conditions - end-user response time (slow network connectivity, number of simultaneous requests)
  - ...

How to measure performance?

- Monitoring local resources (server load)
  - RAM, CPU, harddisk speed, network resources
  - mod_status: basic server statistics (machine-readable)
  - Various open-source and commercial tools, e.g. Nagios, (n)top, mon, mrtg, OpenNMS, ...
- Bechmarking speed
  - Tools: ApacheBench, Flood, httperf, Autobench, ...
  - Increasing number of concurrent requests
  - Pages per second / seconds per request
  - GET, POST, SSL, KeepAlive, simulated user-agents, ...
  - Distributed tests, log replays
  - Repeatable environment (e.g. isolated network, fixed content-length, ...)

Hardware- and OS-dependent decisions

- A lot of RAM – never ever swap!
  - For example use MaxClients to control how many children are forked at maximum
- CPU, HDD, network card – *just fast enough*
- Does your favourite OS support your hardware and the desired features (sendfile, threading libraries, ...)?
- Run the latest stable release and patchlevel of your OS
- Keep your system up to date

Compile-time configuration

- Choose a suitable MPM (`--with-mpm=MPM`)
- Eliminate unused modules (`--disable-`*`module`*)
- Disable DSO support, link modules statically (`-DDYNAMIC_MODULE_LIMIT=0`)
  - This will save RAM that's allocated only for supporting dynamically loaded modules
- Enable a faster atomic compare-and-swap (CAS) implementation (`--enable-nonportable-atomics=yes`)
  - Only useful on SPARC and Linux x86 > 486
- Disable mod_status in production (especially turn off `ExtendedStatus`)

Multi-processing modules

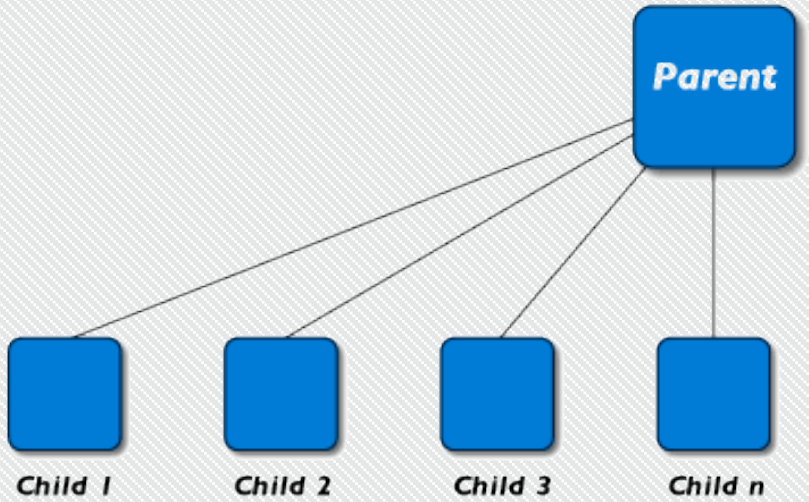
- An MPM defines how the server will receive and manage incoming requests:
  - Different HTTP server process models (e. g. threaded, process-based or hybrid)
  - Platform- & OS-specific optimizations (e.g. Windows, BeOS, NetWare, OS/2)
  - OS-specific features (e.g. AcceptEx, ExceptionHooks, etc.)
  - Admin can choose: Reliability vs. Scalability vs. Performance vs. Features
  - More efficient ways of controlling the server (resource limits, thread/process ratio)
  - Extendable with third-party MPMs

Prefork

- Each child handles one connection at a time: much traffic, many children :)
  - High memory requirements
  - Highly tolerant of faulty modules
  - Highly tolerant of crashing children
  - Fast
  - Well-suited for 1 and 2-CPU systems
  - Tried-and-tested model from Apache 1.3
  - "You'll run out of memory before CPU"

Prefork model

- Each child handles one connection at a time: many children are needed

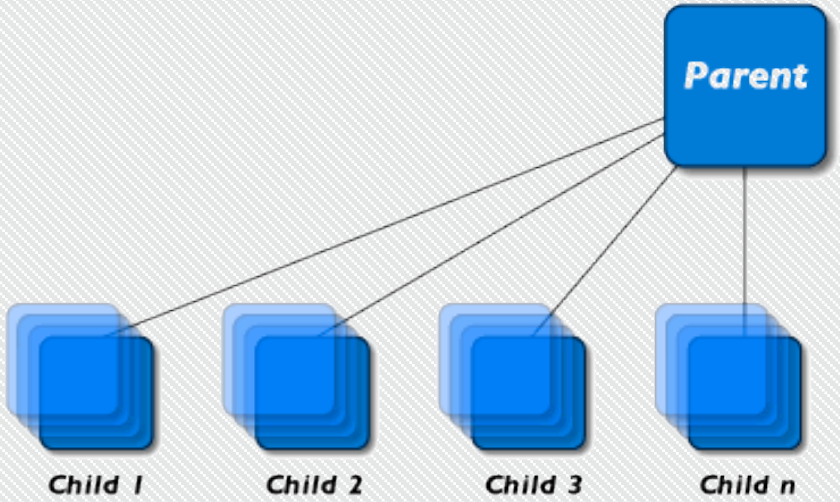


Worker

- Multi-threaded within each child, each thread handles a single connection:
  - Low to moderate memory footprint
  - Moderately tolerant to faulty modules
  - Faulty threads can affect all threads in a child
  - Fast and highly scalable
  - Well-suited for multiple processors
  - Requires a mature threading library (Solaris, AIX, Linux 2.6 and others work well)
  - Memory is no longer the bottleneck

Worker model

- Multi-threaded within each child: only a few children are needed



Other MPMs

- OS-specific MPMs:
  - WinNT
  - OS/2
  - BeOS
  - NetWare
- Perchild (experimental)
- Leader-Follower (experimental)
- Threadpool (experimental)
- Third-party MPMs: Metux-MPM

Choosing an MPM

- Multi-process, multi-threaded, or both?
- Compile-time decision
- Depends on a variety of factors:
  - Does the OS support threads?
  - Scalability vs. Stability?
  - Are third-party modules with unknown, and possibly thread-unsafe extensions (e.g. PHP, mod_perl) used?
  - How much memory is available?
  - ...

Run-time configuration

- Controlling MPM parameters
  - `MaxClients`: big enough to handle the excpected number of requests and small enough to assure that there is enough physical RAM for all processes/threads
  - `ServerLimit`, `ThreadLimit` and `ThreadsPerChild` limit/influence the `MaxClients` setting
  - `ListenBackLog` (increase when under a TCP SYN flood attack)
  - `MaxRequestsPerChild` (decrease when dealing with memory leaks)
  - `ScoreBoardFile` (use anonymous shared memory when possibile or at least a RAM disk)
  - `SendBufferSize` (increase when using high speed pipes)
  - Prefork: MinSpareServers / MaxSpareServers
  - Worker: MinSpareThreads / MaxSpareThreads

Run-time configuration

- Avoid DNS lookups (`HostnameLookups Off`)
  - Use `logresolve` to post-process logfiles
  - Use IP addresses instead of domain names in `Allow from / Deny from` directives
- Disable .htaccess files (`AllowOverride None`)
- Avoid `Options SymLinksIfOwnerMatch` without `Options FollowSymLinks`
  - Apache will have to issue extra system calls to check up on symlinks: `lstat(2)` on every path component
- Avoid content-negotiation
  - Apache has to scan for suitable files
  - use typemaps instead of `Options MultiViews`

Extreme example

- To temporarily defend against DoS attacks or to rescue a server from being slashdotted, the following snippet may be of help:

```
<LocationMatch "^/action.php">
    Order Allow,Deny
    Deny from all
    ErrorDocument 403 "Sorry
    SetEnv nokeepalive
    SetEnv downgrade-1.0
    SetEnv force-response-1.0
</LocationMatch>
```

Know your server

- Keep an eye on the logfiles
- If something goes wrong, the first place to look is always the error_log
- Temporarily increase the LogLevel if needed (debug)
- mod_status shows what Apache is doing
- http://httpd.apache.org/docs-2.0/

Keeping up to date

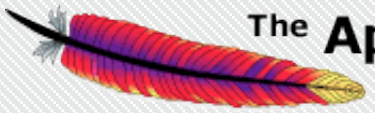
- Apache website and Announcement list
  - http://httpd.apache.org/
  - announce-subscribe@httpd.apache.org
- ApacheWeek
  - http://www.apacheweek.com/
- Vendor package updates
- CERT CC, BugTraq, Full Disclosure List

That's it!

Thanks for listening!

More info and the slides are available at
http://www.apache.org/~erikabele/

You can reach me at
erikabele@apache.org



The Apache Software Foundation
http://www.apache.org/