Sun GlassFish Web Stack Deep Dive

Apache HTTP Server Performance

Jeff Trawick
Sun GlassFish Web Stack, Sun Microsystems

What role can Apache play in the performance of the overall system

Alternatives (Lighttpd, Sun Web Server)

Tuning for capacity and performance

Agenda

- Why Apache
- Key choices to make
- Configuring Apache for capacity
- Configuring Apache for performance
- For more information

Why Apache

- First: Don't choose Apache over other web servers for performance reasons
- Consider also Sun Web Server and Lighttpd
  - Both use fewer system resources per client than Apache
  - Both can provide higher response rates than Apache
  - Both of these, in addition to Apache, are covered by GlassFish Portfolio support subscriptions

Why Apache

- If it is bloated and slow, why does it still matter?
  - Rich set of features, along with configurability
  - Multitudes of plug-in modules available from third parties
  - Extensive documentation
    - From the project itself
    - From the community
    - From traditional publishers
  - Multiple vendors investing in it
    - Sun, SpringSource, IBM, RedHat, etc.
  - Many people skilled in configuration/tuning
  - Analyzed extensively for security flaws
  - Long history of addressing security issues in a responsible manner

Why Apache

- Oh, and it can perform adequately in most circumstances

Why Apache

- From this point forward, we'll assume there's a good reason to use Apache

Key choices to make

- What Apache features can improve overall system performance

- Process model
  - Prefork vs. worker MPM
  - FastCGI vs. in-process script execution

Using Apache to improve application performance

- Not Apache tuning *per se*
- What Apache features can I use to improve overall application performance
  - Whether or not it makes Apache slower or makes Apache consume more resources

Using Apache to improve application performance

- Several common areas:
  - Cache
  - Compression
  - Load-balancing to back-end servers

Using Apache to improve application performance - Cache

- Manipulating cache headers for use by client and network caches
  - If application doesn't generate those
- Apache's cache
  - Request still reaches Apache but is served from static file on disk (VM) instead of forcing application to render again

Adding Expires headers

- If the application doesn't set Expires headers so that the browser (or cache) knows how long the response is valid, Apache can add these via mod_expires
  - Uh, only if the resource won't change for the configured time, or the browser doesn't actually need to see the changed version for the configured time

Adding Expires headers

- Here's what it looks like:

```
<Location /my-status-app/>
 ExpiresActive On
 # The response is valid for the next hour.  Browsers
 # don't have to request it again.
 ExpiresDefault "access plus 60 minutes"

 </Location>
```

Any of Apache's configuration containers can be used for this.

Adding Expires headers

- Special concern:
  - Once the response is sent, it can affect not just the client that requested it but also network caches.
  - If you really really need to start serving an updated version of the resource, some clients will just have to wait.

Apache's cache

- mod_disk_cache is the only actively developed cache bundled with Apache
  - mod_mem_cache probably *needs* active development
  - Other, simpler caching is of more limited use (e.g., mod_file_cache)
- Typically used for dynamic content
- Also useful when data is stored on slow (network?) disks and the cache can be kept on fast local disks
- You still need cache information from the application (or added by Apache as in the previous mod_expires example).

mod_disk_cache

- If you're using it to avoid requests to the application, you can get value from it with complete control over caching (i.e., retain ability to start sending new content immediately):
  - Use mod_expires to add the Expires headers so that it is cacheable (locally)
  - Remove this information with mod_headers before the response is generated.
  - If you have to start sending new content immediately, it is only cached locally via mod_disk_cache, so remove the cache files with htcacheclean.

mod_disk_cache – for more information

- http://blogs.sun.com/trawick/entry/caching_dynamic_content_with_apache

Using Apache to improve application performance - Compression

- Much of web traffic is highly compressible (HTML, JavaScript, style sheets)
- Spending CPU in the web server to compress output can significantly improve user experience
- Use mod_deflate's DEFLATE filter
  - Simple to configure; check the Apache docs at

    http://httpd.apache.org/docs/2.2/mod/mod_deflate.html

Using Apache to horizontally scale applications

- Reverse proxy to the application server (GlassFish, Tomcat, Mongrel, etc.)
- mod_jk is a traditional implementation
  - Well documented for use with Tomcat
  - Use with GlassFish is described in a number of blog articles
- Apache 2.2 brings
  - mod_proxy_ajp (AJP protocol, like mod_jk)
  - mod_proxy_balancer (balancer supports AJP protocol like mod_jk, but also supports traditional HTTP proxy too, for use with Mongrel or anything else)

More on mod_proxy_balancer

- Here's a detailed article on using mod_proxy_balancer, plain HTTP proxy, and Mongrel (for Rails):

  http://blog.innerewut.de/2006/4/21/scaling-rails-with-apache-2-2-mod_proxy_balancer-and-mongrel

Process model

- Prefork vs. worker MPM

Prefork MPM

- One single-threaded process per active connection
  - Matches Apache 1.3 processing model
- Most commonly used
  - Prefork is the default or only MPM provided by a number of vendors
- Compatibility with the most modules
  - Recommended for mod_php
  - Required for mod_perl if Perl interpreter doesn't support threads
- Avoids concurrency bugs in plug-in modules, or concurrency-related performance issues in libraries
- More limited damage from vulnerabilities or crashes
  - Complete isolation of client requests

Prefork MPM - drawbacks

- Uses the most resources
  - Requires a surprising amount of swap space on Solaris
  - Larger working set (physical memory)
- Poor utilization of per-process state, such as
  - Retained connections to other servers, such as
    - Java application server
    - LDAP server
  - Process-memory caches
- Requires more difficult shared memory + cross-process synchronization to effectively share computed information (e.g., APC)
  - So often not implemented

Worker MPM

- New with Apache 2.0
  - Extensively used in production environments since 2002
- Minimizes memory consumption
  - Unlike prefork, doesn't require surprising amounts of swap space on Solaris
  - Smaller working set
- Can allow effective utilization of retained state without more-difficult shared memory implementation
  - Increasing effectiveness with lower number of child process

Worker MPM, retained state

- In-memory caches
- Server connections
  - Application server
  - Database connection
  - LDAP server connection
  - All can be much better utilized by using worker instead of prefork
  - Ouch! if a connection retained by a prefork process (in hopes of handling a request again soon) consumes significant server resources

Worker MPM - drawbacks

- Occasional issues with releasing resources as child processes exit (more later)
- Maximum exposure to vulnerabilities and other bugs
  - Information for other users could be retrieved
  - If processing for one user triggers a crash, other users are impacted

    (can be mitigated somewhat by reducing the number of threads per child)

Tuning for capacity

- What does this cover?
  - How many clients can I support
  - How much of my system can Apache use?

Tuning for capacity

- How many clients can I support?
  - This is relatively simple (essentially MaxClients).

    (more later on that)

Tuning for capacity

- How much of my system can Apache use?
  - This is quite important, but Apache puts all the work on the system administrator
  - Administrator must set MaxClients low enough to avoid running out of
    - Swap space
    - Physical memory

    yet high enough to serve enough clients.
  - Observation required for new Apache or application deployments (mod_status, vmstat, etc.)

Tuning for capacity

- Special Solaris issue
  - Solaris allocates swap space based on the virtual memory size of each process, including code that is shared among the Apache processes
    - Removing code lowers the swap requirements (times number of child processes)
  - Comment out any unnecessary LoadModule directives before working on capacity
  - If using in-process PHP (mod_php), disable any unused extensions as well.

Tuning for capacity - Keep-Alive

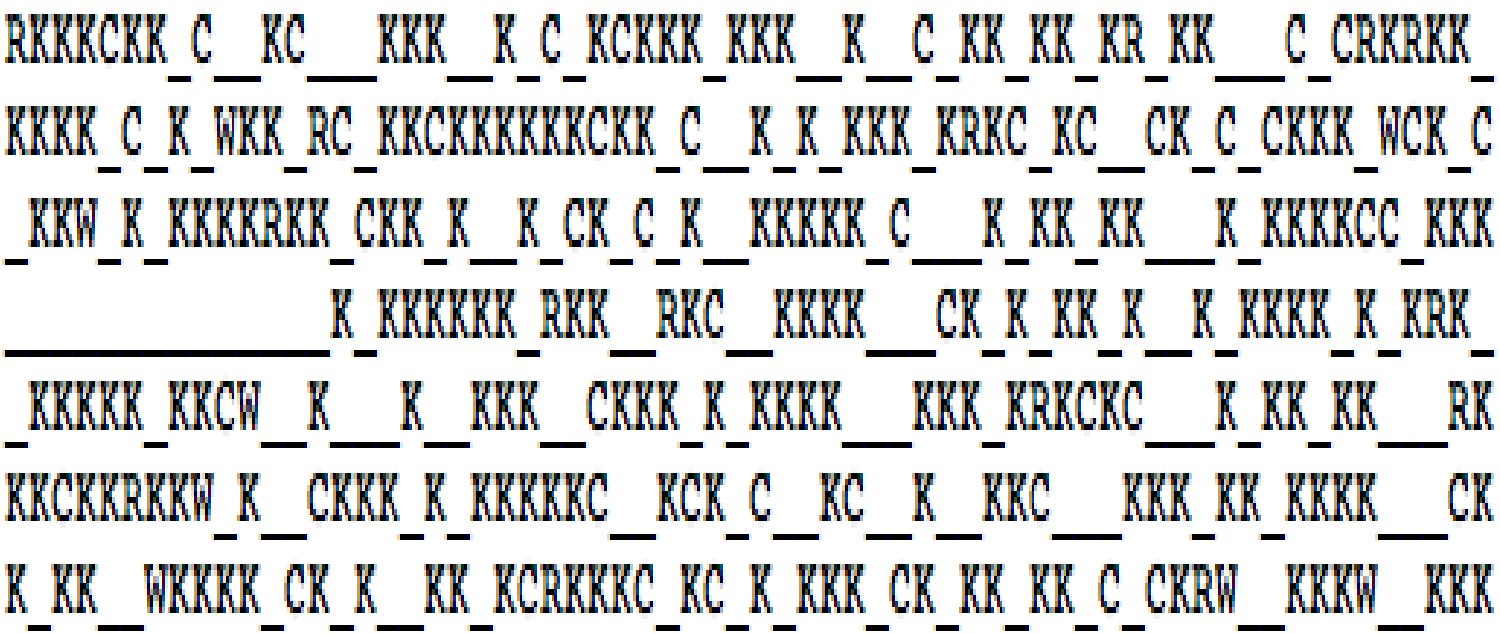
- KeepAlive {On|Off}
- KeepAliveTimeout [timeout-in-seconds]

- Because Apache* dedicates a processing thread to connections in keep-alive state, this is a capacity concern in addition to a performance-tuning issue

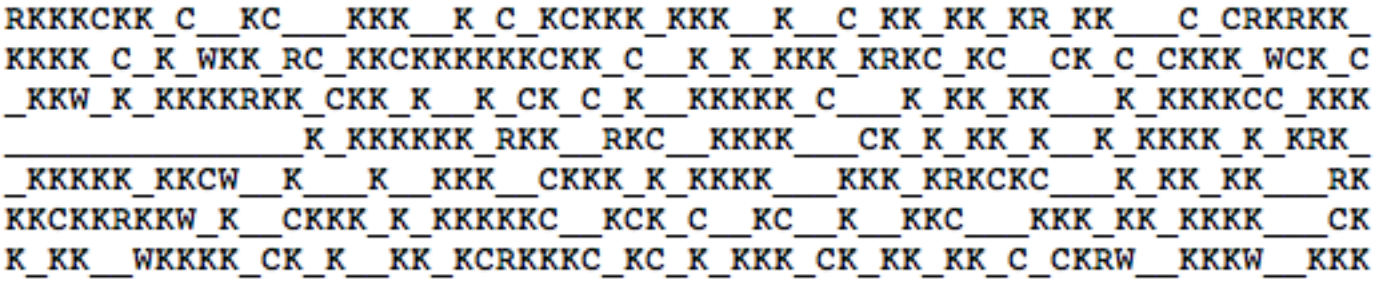
- Reduce the number of processes (prefork) or threads (worker) required by reducing KeepAliveTimeout to a small value (e.g., 2 seconds)

  *The Event MPM handles Keep-Alive state without a dedicated thread. (Event as default MPM for 2.4?)

Tuning for capacity - Keep-Alive



What does this mean?



- If we need to increase the number of clients we can handle, and/or reduce system resources:
  - Reduce KeepAliveTimeout

Tuning for capacity

- Capacity of Apache vs. capacity of back-end servers
  - There's almost never a 1:1 mapping between Apache and a back-end server
    - Either Apache off-loads certain types of requests or loadbalances to multiple back-ends (*usually*)
      - Apache will be handling more clients than a particular back-end server
  - Be aware that increasing Apache's capacity beyond that of back-end server(s) can use extra system resources without much benefit (under heavy load and/or when the application isn't responding)
    - Example trade-off: Browser reports that the site is unavailable, vs. application renders a page indicating that there are too many database connections in use (the latter exacerbating the capacity problem)

Cleaning up Apache child processes (Capacity?)

- "Set MaxSpareServers/MaxSpareThreads kind-of low so Apache doesn't use so many processes/memory/whatever"
  - What's missing here? Apache essentially "owns" or has reserved enough system resources to run at maximum capacity 24x7.
  - If user load increases or an application slows down, Apache can very rapidly reclaim the resources. In other words, you can't use them for anything else.
  - But there's a valuable side-effect to idle process termination that affects other servers: Any retained connections (app server, DB, LDAP) will be cleaned.
    - Very important with prefork, since such connections are so poorly utilized to begin with. (not so important with worker unless ThreadsPerChild is low)

Cleaning up Apache child processes

- How?
  - MaxSpareServers (prefork), MaxSpareThreads (worker)
    - Reduce process count when load subsides
  - MaxRequestsPerChild
    - Special purpose: work around memory or other resource leak
  - Graceful restart

Worker MPM and child process exit

- Maybe it is helpful for us to terminate excess processes (e.g., to terminate retained back-end connections)
- Worker has a special problem:
  - A child can't exit (i.e., *release resources*) until the last client connection handled by the process has finished.
  - The occasional hung or very long running request is not uncommon in some environments.
    - Worst case: Because of trying to keep the process count low in presence of hung requests, you can accumulate a huge amount of processes with only 1 or 2 active threads. The bulk of system resources is still used.
- Ouch! You need to use MaxRequestsPerChild to work around a leak, but a small number of requests take a very long time.

Prefork MPM and child process exit

- A child process only handles one client connection (if not idle), so no special issues surrounding child process exit

(well, third-party modules can create some havoc in their child exit handlers, but I haven't seen that in a while)

Tuning for performance

- What does this cover?
  - After establishing the big picture (what role does Apache play, what MPM is used), minimize system resources used and/or improve response times.

Tuning for performance – Prefork MPM

- StartServers, MinSpareServers, MaxSpareServers
- StartServers can speed up the rate at which Apache can start handling high load (useful if behind a load balancer which can suddenly divert a lot of traffic)
- If StartServers > MinSpareServers and there's no initial burst of traffic, Apache will create a bunch of children at startup then start terminating them (waste)
- If you don't get a burst of traffic initially, don't worry about StartServers (but don't set it real high).

Tuning for performance – Prefork MPM

- Keep in mind the proportions of MinSpareServers and MaxSpareServers to MaxClients
- Some people scale up MaxClients drastically, but don't scale up MinSpareServers/MaxSpareServers.
  - Apache ends up terminating/creating child process when the server load changes by a very small amount
  - Consider MaxSpareServers 100 and MaxClients 8192
    - If load decreases by just over 1% of the overal max, Apache will terminate child processes (and likely have to create more soon after).

Tuning for performance – Prefork MPM

- Is this bad?

  MaxClients 1024
  MinSpareServers 256
  MaxSpareServers 256

- Yes; constant termination/creation of child processes

- Mind the gap (between Min and Max).

Tuning for performance – Prefork MPM

- What about ServerLimit?
  - Most people can sleep well not knowing the details and simply setting ServerLimit = MaxClients.
  - ServerLimit is the number of process slots in the Apache scoreboard, which can't be reallocated across graceful restart.
  - Thus, if you want to change MaxClients across graceful restart, you can't make it higher than ServerLimit (and you can't change ServerLimit).

Tuning for performance – Worker MPM

- Much the same as prefork, but with an added dimension – threads per child.

- Higher ThreadsPerChild:
  - Lower overall memory use
  - Better utilization of per-process resources like reusable back-end connections
  - (Potentially) Higher thread contention
  - Higher risk of information leaking from another thread
  - More clients impacted if one client triggers a crash

Tuning for performance – Worker MPM

- Special file descriptor-based issues with higher ThreadsPerChild
  - Traditionally some third-party modules used select() on back-end connections, and blew up with fds > 1024
    - Limit ThreadsPerChild to 500 or so
      - 500 client connection sockets
      - 500 backend connection sockets
      - Log files, listening sockets, etc.
  - Traditionally some third-party modules or libraries used used fopen() on Solaris failed unless a free fd under 256 was available.
    - Web Stack's Apache delivery resolves that

Tuning for performance – Worker MPM

- Similar to prefork MPM, but “Thread” instead of “Server”
  - Pay attention to the proportion of MinSpareThreads and MaxSpareThreads to MaxClients, and to the difference between them
  - MaxClients 8192, MinSpareThreads 96, MaxSpareThreads 192
    - Probably means endless work
  - For MaxSpareThreads (cleaning up extra processes), remember worker's special issue when some occasional requests are long-running

Tuning for performance

- Other usual suspects
  - Avoid DNS lookups
    - HostnameLookups Off
    - Allow/Deny from IP address (instead of hostname)
  - Don't force Apache to look for .htaccess files in every directory down to the file
    - AllowOverride None
    - If you must use .htaccess, allow in minimal set of directories.
  - Don't load modules you don't need
  - Avoid excessive logging
  - Don't force Apache to check if parts of the path down to the file are symlinks
    - Use option FollowSymLinks
    - Don't use option SymlinksIfOwnerMatch (what about about untrusted users?)

Tuning for performance

- Concerns with untrusted users managing part of the web space
  - You don't want Apache blindly following symlinks
    /export/home/joeuser/public_html/index.htm → /etc/xxx
    - Disabling FollowSymLinks will force Apache to use lstat() on each directory/file under the user's control
  - They can't control httpd.conf, but maybe they need to control mod_autoindex or mod_rewrite or XXX
    - The AllowOverride will force Apache to look for .htaccess files

Memory-related tuning

- ThreadStackSize
  - Worker MPM only, and only for worker threads
    - use ulimit for prefork stack size or worker main/listener threads
  - On Solaris, it is rare that you need to *increase* it
  - You can decrease it from the default
  - A good place to start: ThreadStackSize 131072

Memory-related tuning

- Apache retains memory obtained during request processing to be used on future requests on the same thread
- If some relatively infrequent requests consume significantly more memory than usual, releasing this extra memory at the end of the connection can lower overall process memory usage
- MaxMemFree number-of-kbytes
  - Release any additional memory beyond this amount back to the heap
- Most effective with worker, since memory released to the heap can be utilized by another thread

KeepAlive and KeepAliveTimeout

- Discussed previously because of its capacity ramifications
- Important to have KeepAlive On because of typical application behavior with client fetching images, stylesheets, and javascript files on the same connection
- Decrease KeepAliveTimeout to a few seconds to get the most benefit without using too many processes/threads

KeepAlive and KeepAliveTimeout

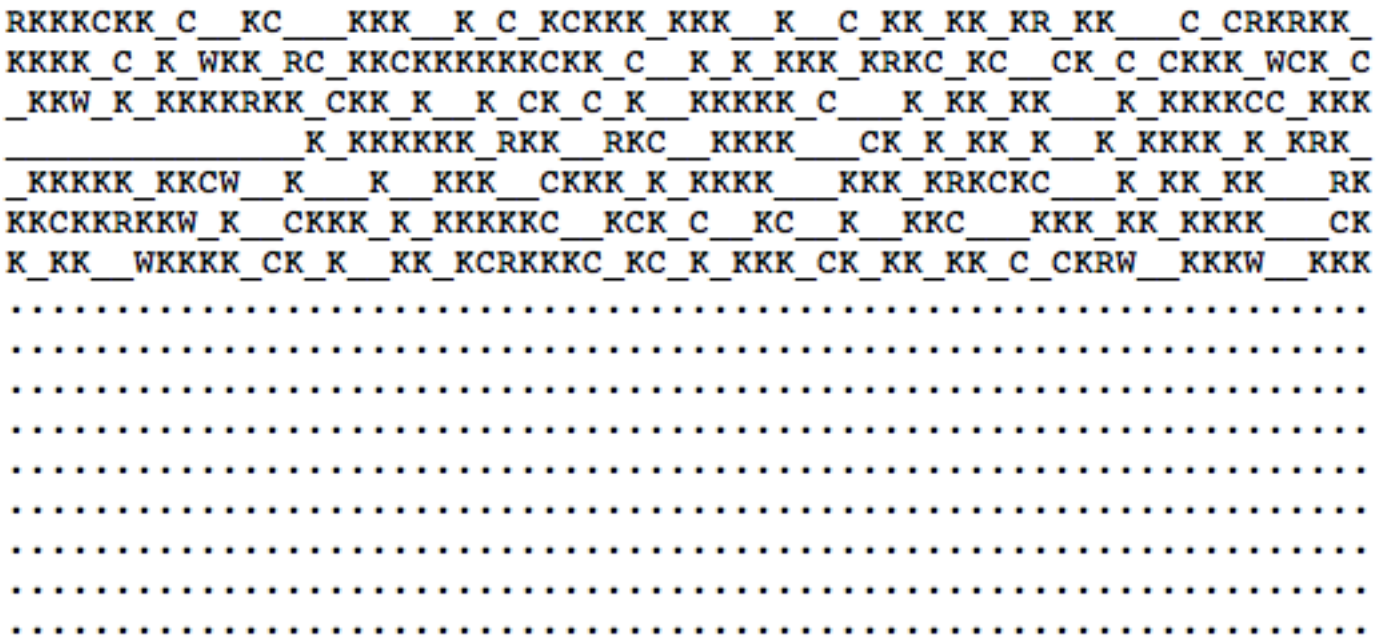
- SSL-enabled virtual hosts
  - A higher KeepAliveTimeout can be beneficial here, since extra processing is required even when an existing SSL session is resumed.
    - Not effectively resuming (reusing) SSL sessions? Fix that first.

ExtendedStatus

- ExtendedStatus is a directive of mod_status
- When On, Apache core saves more information in the shared-memory scoreboard for use (display) in the mod_status report (or retrieval by some third-party modules)
- As is often noted, it does increase CPU utilization
  - Typically < 1%
  - The detailed status reports are important for debugging
  - Set ExtendedStatus On and don't worry about the performance hit (assuming you use mod_status reports).

ExtendedStatus

- Information available with ExtendedStatus Off



ExtendedStatus

- Information available with ExtendedStatus On
  - The map on the previous slide, plus

Srv	PID	Acc	M	CPU	SS	Req	Conn	Child	Slot	Client	VHost	Re
0-54	25189	0/104/789374	R	33.94	32	2	0.0	0.39	165512.81	?	?	..reading..
0-54	25189	1/82/792723	K	45.65	2	2	6.3	7.87	165845.25	64.0.193.26	httpd.apache.org	GET /docs/2.0/images/feather.gif HTTF
0-54	25189	1/199/795400	K	45.44	4	75	9.2	1.57	164337.11	116.48.4.7	archive.apache.org	GET /dist/ HTTP/1.1
0-54	25189	1/99/797657	K	45.36	4	2	0.6	3.41	162550.19	65.55.106.206	beehive.apache.org	GET /docs/1.0/apidocs/classref_netui/in
0-54	25189	1/197/800017	C	45.22	1	3	1.6	0.50	164994.16	117.32.175.233	commons.apache.org	GET /dbcp/guide/index.html HTTP/1.1
0-54	25189	1/82/795696	K	45.46	3	226	0.1	0.40	163812.33	92.103.136.180	httpd.apache.org	GET /docs/2.2/images/down.gif HTTP/
0-54	25189	2/73/800799	K	45.69	2	2	9.3	5.72	166740.20	15.203.233.79	incubator.apache.org	GET /images/apache-incubator-logo.pn
0-54	25189	0/91/799160	_	43.37	17	7	0.0	3.70	169419.78	115.102.39.34	archive.apache.org	GET /dist/struts/binaries/struts-2.0.12-al
0-54	25189	1/91/806100	C	45.24	1	10	6.0	1.14	163802.28	62.177.71.161	geronimo.apache.org	GET /style/default.css HTTP/1.1
0-	25189	0/86/809149		44.49	11	3	0.0	5.26	171098.64	61.247.222.55	activemq.apache.org	GET /robots.txt HTTP/1.1

SSI (Server Side Includes)

- This type of document requires expensive processing to parse and evaluate SSI tags (e.g., for including a header or footer)
  - mod_include implements SSI for Apache
- Many examples show SSI documents with a special file extension – .shtml instead of .html
- What if you don't want to expose that odd extension?
  - Naïve implementation: Just pass all .html files through mod_include
    - Needless processing for non-SSI files
  - Better performance: Set XbitHack On and run “chmod u+x” on just the files that have SSI tags
    - No parsing of non-SSI files

Sendfile*v*

- The idea: let the kernel do more of the work to send file contents to the client
  - Fewer syscalls (no explicit read of file or write of socket)
    - Can be fewer user space dispatches too
  - VM magic
- Controlled by EnableSendfile {On|Off}
- Currently disabled in Web Stack's Apache delivery
  - Past system issues with sendfilev
  - Past performance issues
- Need to revisit
  - A lot of sendfile work went into 2009.06

Mmap-ed files

- The idea: let the application reference file contents like memory
  - Fewer syscalls (no explicit read of file; still must write to socket)
  - VM magic
- Apache generally uses mmap-ing when reading files, such as for SSI processing or just sending the contents as-is (when sendfile isn't used)
- In a number of cases, performance has improved on Solaris after disabling this (EnableMMap Off)

TCP-related tuning

- SendBufferSize
  - Solaris defaults to a fairly large value (1MB?), which is usually sufficient
  - Large values prevent Apache from waking up often to pass more data to the TCP layer (as the client ACKs previous data)
- This setting specific to Apache avoids needing to change the global Solaris default, which is often recommended.

TCP-related tuning

- ReceiveBufferSize
  - Same as SendBufferSize, but for large request bodies sent by clients (e.g., uploaded files)
- This setting specific to Apache avoids needing to change the global Solaris default, which is often recommended.

TCP-related tuning

- tcp_conn_req_max_q0, tcp_conn_req_max_q, and ListenBacklog
- Don't bother setting a high ListenBacklog without also adjusting tcp_conn_req_max_q (and don't bother setting that high without setting _q0 at least as high).
  - Jeff's recommendation: Leave these alone until you see connections getting dropped.
    - Run 'netstat –s | fgrep –i listendrop'. If tcpListenDrop is non-zero, increase tcp_conn_req_max_q. If tcpListenDropQ0 is non-zero, increase tcp_conn_req_max_q0.

http://blogs.sun.com/terrygardner/entry/solaris_tcp_ip_parameters_tcp

Further topics to explore

- Effects on performance when switching from in-process interpreter (e.g., mod_php) to FastCGI mode
  - FastCGI apps can be multi-instance but don't handle concurrent connections, thus avoiding potential problems with the worker MPM, and thus allowing system resource use to be decreased via MPM choice
- SSL tuning, including Solaris' Kernel SSL

For more information

- Apache's performance tuning guide
  http://httpd.apache.org/docs/2.2/misc/perf-tuning.html

  (But some out of date information, or hints for build-time tweaks)

- Yahoo's “Best Practices for Speeding up your Web Site”
  http://developer.yahoo.com/performance/rules.html

  (Mostly in the application domain)

Sun GlassFish Web Stack Deep Dive

Jeff Trawick
Jeffrey.Trawick@Sun.com