Apache HTTP Server 2.4 Problem Diagnosis

Jeff Trawick

November 6, 2012

Table of Contents

Introduction — Who am I?

- Since 2000 I've worked on Apache httpd and APR.
- A lot of that time I had customer support responsibilities.
- I've been very interested in patches to httpd or new httpd capabilities which make it easier to diagnose problems.
- This year I've been able to revisit some earlier efforts on httpd diagnosability; I'll mention a few of these projects (or *experiments*) during this talk.

Introduction — What will we attempt to cover?

- Cover the basics in a reasonably broad manner, but when in doubt be sure to touch on new httpd 2.4 features since knowledge of basic httpd debugging techniques is prevalent.
- Summarize the techniques which are different with httpd 2.2.
- Conclude with some examples showing how debugging issues with nginx can differ considerably.

What kinds of issues encountered

- Crash
- Hang of server
- Stall of individual requests
- Termination
- Bad response time
- Limited concurrency without problem symptoms
- High CPU
- High memory
- High consumption of other pooled resources
- Incorrect output - wrong transformation
- Incorrect output - missing/bad protocol element

Non-problems

- Validate behavior of new software/configuration
- Understand steady-state behavior for baseline when something is wrong

Using tools inside the web server

- Logging (the information itself, the timestamp, information about other processing at about the same time)
- OS-level tools (view use of resources, whether discrete items like files or continuous like
- CPU-, code-level tools (determine what code is running frequently, what is running for the request, analyze memory references, walk through the processing of a request, etc.)

Careful with logging!

As you increase the level of logging, you increase the chances that private data will be logged.

- Passwords
- Session keys
- ???

Of particular interest:

- mod_dumpio, mod_log_config when configured to log certain request or response header files, mod_log_forensic, http when configured at higher trace levels, ...

Logging

- Error log

Error log records

- Configurable content
- Fields dropped when information is unavailable
- Third-party modules can implement their own fields

Typical message:

```
[Sun Oct 28 13:37:27.676386 2012] \
[-:error] \
[pid 14340:tid 140625844377344] \
[client 127.0.0.1:50837] \
mod_wsgi (pid=14340): Target WSGI script \
'/home/trawick/myhg/apache/documents/AC20\
12EU/lookup.wsgi' does not contain WSGI a\
pplication 'application'.
```

Detailed logging only for specified client IP

```
LogLevel info
<Location />
<If "%{REMOTE_ADDR} =~ /127.0.0.0/">
LogLevel trace8
</If>
</Location>
```

- Only works once request processing has reached a certain point. Connection-level issues which occur before that point won't be logged.
- The unexpected `Location` container works around a bug which `sf` may have fixed since yesterday.

Detailed logging for problematic requests

```
LogLevel info
<Location /problem/>
  LogLevel trace8
</Location>
```

(Mostly) HTTP layer logging at different levels

```
[core:trace5] Request received from client: GET / HTTP/1.1
[http:trace4] Headers received from client:
[http:trace4]   Connection: keep-alive
[http:trace4]   Cache-Control: max-age=0
[http:trace4]   User-Agent: Mozilla/5.0 (X11; Linux x86_64
[http:trace4]   Accept: text/html,application/xhtml+xml,ap
[http:trace4]   Accept-Encoding: gzip,deflate,sdch
[http:trace4]   Accept-Language: en-US,en;q=0.8
[http:trace4]   Accept-Charset: ISO-8859-1,utf-8;q=0.7,*;q
[http:trace4]   If-None-Match: \\"2d-4b1922bade1c0\\"
[http:trace4]   If-Modified-Since: Sat, 12 Nov 2011 23:41:
[http:trace3]  Response sent with status 304, headers:
[http:trace5]   Date: Tue, 06 Nov 2012 12:18:57 GMT
[http:trace5]   Server: Apache/2.4.4-dev (Unix) OpenSSL/1.
```

mod_log_debug

- Configurable debug logging mechanism using new `LogMessage` directive.
- Different ways to think of it:
  - Generate custom trace or error messages for processing of interest to you.
  - Track interesting values as they change (or not) during request processing.
- Conditional expression support with access to dynamic values is provided by the new *ap_expr* support.
  - `http://httpd.apache.org/docs/2.4/expr.html`

mod_log_debug – sample configuration

```
# Log some module’s request note at all phases
# of processing (but only if set)
<Location />
  LogMessage "%{note:mod_your_debug}" hook=all \
    "expr=-T %{note:mod_your_debug}"
</Location>
# Log when a location is requested as a subrequest
<Location /dir/file1/>
  LogMessage "subrequest to /dir/file1/" \
    hook=type_checker "expr=-T %{IS_SUBREQ}"
</Location>
# Log when a particular error is encountered
LogMessage "Timeout from %{REMOTE_ADDR}" \
    "expr=%{REQUEST_STATUS} = 408"
```

mod_dump_io

- This is a way to trace the raw, unencrypted data exchange into the error log.
- A packet trace is often preferable (when feasible), but this module is trivial to use as long as there aren't production environment issues.

mod_dump_io output

```
LogLevel info dumpio:trace7
DumpIOInput On
DumpIOOutput On
```

mod_dump_io output

(extraneous info removed in order to fit)

```
dumpio_in [getline-blocking] 0 readbytes
dumpio_in (data-HEAP): 20 bytes
dumpio_in (data-HEAP): GET /dir/ HTTP/1.1\r\n
dumpio_in [getline-blocking] 0 readbytes
dumpio_in (data-HEAP): 22 bytes
...
dumpio_in (data-HEAP): Connection: keep-alive\r\n
```

Catching requests which do not finish

```
LoadModule log_forensic_module modules/mod_log_forensic.so
ForensicLog logs/forensic.log
```

This logs the start and end of the request along with all of the request headers.

```
+UJggYn8AAQEAAAs1da4AAAAA|GET / HTTP/1.1|Host...
-UJggYn8AAQEAAAs1da4AAAAA
```

- `check_forensic` will scan the log and determine which requests didn't finish cleanly.
- Compare with mod_whatkilledus, described later.

Where did that error message come from?

- module id in error log:

```
[core:info] [pid 4373:tid 140043736946432] ... AH00128
```

- whoops, missing module id (needs a minor tweak for httpd 2.4):

```
... [-:info] [pid 8889:tid 140363200112416] mod_wsgi (
```

(We know the module because it includes it in the message, but there's no guarantee of that.)

```
... [-:error] [pid 14883:tid 140625458312960] 1
... [-:error] [pid 14883:tid 140625458312960] 2
... [-:error] [pid 14883:tid 140625458312960] 3
... [-:error] [pid 14883:tid 140625458312960] 4
```

(I think that might be mod_wsgi logging stderr from a script????? I don't remember what I was doing at the time.)

Warning! Gratituitous plug for weird feature!

Sick feature that might be useful in cases where no module id is available or the logger is a utility function:

- mod_backtrace has the capability of adding a backtrace to error log messages in certain conditions.

  ```
  ErrorLogFormat ... [%{/AH00128/}B] ...
  ```

- If the search string appears in the message, a mini-backtrace will appear as an additional field in the error log record.

  ```
  ... [0x7f75aaa7c6a4<0x7f75aaa7c962<0x45993a<0x45 \
  a096<0x442f6f] ... AH00128:...
  ```

  (highly dependent on symbols and OS-specific backtrace APIs)

- `http://emptyhammock.com/projects/httpd/diag/`

Looking from the outside

- examining resource use
- tracing activity

System call trace

- strace/truss/dtruss
- FreeBSD: fstat, procstat, ss, smem

Looking inside the process with a debugger

Basic information: Backtraces

- gdb
  - Most platforms (even Windows, using MinGW gdb on MinGW build of httpd)
  - Basic use:

    ```
    gdb /path/to/httpd pid-or-corefile
    (gdb) thread apply all bt full
    ```

    (but other commands may be useful too)
- pstack
  - Solaris (I learned through bad experiences to pretend that pstack isn't available on Linux)
  - Use:

    ```
    pstack pid-or-corefile
    ```

    (but pflags and pldd information is also good)

Getting more debugging information

- The backtraces (with variables if available) are most important, but more information is available if you ask for it.
- gdb, more details:

```
(gdb) info sharedlibrary
(gdb) info threads
(gdb) thread apply all bt full
(gdb) thread apply all x/i $pc
```

- Solaris /proc tools:

```
# pstack 13579
# pldd 13579
# pflags 13579
```

Example output

Jeff, this is where you show ubuntu64.core.collect.gdbout and solaris10.core.pstackout.

Umm, what do you do with that?

- Recognize normal behavior
- Determine where crash likely occurred
- Determine definitively where crash occurred

(similar issues for hang)

Umm, what does that stuff mean? (cont.)

- Perplexing (?) problem: Show that output to an httpd developer and they can quickly determine the important parts (i.e., pick the interesting thread)
  - or determine that there's nothing interesting, which can be just as important
- Users typically report the least interesting thread from the coredump, which wastes there time and ours.
- Some sort of automatic annotation/explanation would be useful.
  - Descriptions of normal activity
  - Bug numbers for backtraces that match known problems
  - *et cetera*

Demo

Jeff, this is where you go to http://emptyhammock.com/projects/httpd/explore/. Try loading PR36497.gdbout, PR53870.pstackout, ubuntu64.core.collect.gdbout).

What if you build the code differently

- Improving general debuggability of the generated code by affecting code generation or symbols
- Enabling optional run-time checks
- Enabling third-party exception hooks
- Enabling third-party tracing of API hooks

Different code generation for debugging

- Adding symbols, not stripping executable
- Disabling in-lining of functions for better diagnosablity
- Disabling other optimization so that more variables can be checked

(huge YMMV)

General debug capabilities not built in by default

- Hook tracing
- DTrace probes in the server (DTrace provider *ap*)
- Exception hooks

Hook tracing

- httpd hooks are what allow different modules to handle or otherwise affect processing of the different phases of execution.
- A module that needs to take part in a particular aspect of connection or request processing uses a special hook macro to save a callback pointer.
- At the point where httpd core passes control to modules, it invokes a special hook macro to continue calling module callbacks until a failure occurs, a module elects to handle the request, or all callbacks have been serviced (depending on the hook).
- By tracing what happens inside the hook invocation, some types of failures can be quickly tracked to a particular module.

Hook tracing (cont.)

- httpd now provides a way for third-party code to run during the hook macros at the following points:
  - Start of the hook execution
  - About to call a particular module's hook function
  - Returned from that module's hook function
  - End of the hook execution
- Code inserted into the calling of different modules' handler functions can determine what module's handler took ownership of this phase of request processing and/or caused the request to fail.
- More generally, if some mysterious error occurs at any phase of processing, such as the notorius 500 with no log message, hook tracing could pinpoint the module.

Enabling hook tracing

- Configure argument `--enable-hook-probes` causes `ap_hook_probes.h` to be included in files with hook definitions, making special macros active.
- `ap_hook_probes.h` isn't part of httpd, so it needs to be copied into `include` or located via `CPPFLAGS`.
- Any code invoked by the macros in `ap_hook_probes.h` has to be compiled into the server, so this can be handled by statically linking a module into the server if it won't *fit* in macro form.

Enabling hook tracing (cont.)

- Build mechanism for including this code

```
export CPPFLAGS=-I/path/to/module
./configure --enable-hook-probes \
--with-module=debugging:/path/to/module/mod_foo.c \
--other-args
```

- After httpd is built, `httpd -l` will show `mod_foo.c` as built-in (like `core.c` and a few others).

mod_hook_ar — Experimental hook tracer

- Must be built into the server as with other hook trace code.
- Sets a request note to information about the active module while a hook is active.
- Sets a request note to information about the failing module if a hook returns an error.
- Logging the `RequestFailer` note in the access log:

```
127.0.0.1 ..."GET /cgi-bin/printenva" \
    404 215 mod_cgid.c/404/handler
```

- Can log the name of the `ActiveModule` note in the case of a crash:

```
... [pid 30568:tid 140369329334016] Crash state: \
    mod_crash.c/handler
```

- Download from `http://emptyhammock.com/downloads/`

Possible directions with hook tracers

- How much performance degradation?
- Can this be used to implement DTrace probes?
- Can a built-in module provide a simple API for loadable hook debug modules?
- Will someone write a script to help with generating the right set of macros based on the hooks that need to be instrumented?

(if indeed this is interesting to anyone)

DTrace probes

- httpd-specific probes enabled via `--enable-dtrace` was the goal for 2.4, but only part of the code was committed, and it hasn't been kept up to date with new hooks.
- Someone needs to take interest in getting it working on one of the several platforms with DTrace.
- Existing DTrace providers can certainly help understand httpd processing.
- The pid provider provides great info but it is problematic with httpd because you have to specify a particular process id.

Exception hooks

- `sig_coredump()` is the handler for fatal signals with httpd on Unix since the httpd 1.3 days.
- It changes to the configured core dump directory and rethrows the signal, causing the process to exit and (possibly) the system to create a core file.
- If the `--enable-exception-hook` configure option was specified, `sig_coredump()` will also call exception hooks.
- This allows third-party modules to clean up some resource or save diagnostic information in the event of a crash.

Example exception hook module — mod_whatkilledus

- Like mod_log_forensic, this module saves information about the client request in an early request processing hook.
- Unlike mod_log_forensic, the it is kept in memory during the life of the request, and only logged if a crash occurs.
- Also, if mod_backtrace is loaded it will capture a backtrace for the crashing thread.

mod_whatkilledus report

```
**** Crash at 2012-09-06 14:48:23
Process id:  23368
Fatal signal: 11

...
/home/trawick/inst/24-64/bin/httpd:ap_run_fatal_exception+
...
/home/trawick/inst/24-64/modules/mod_crash.so:0x7fecbd59e9
/home/trawick/inst/24-64/modules/mod_crash.so:0x7fecbd59ea
/home/trawick/inst/24-64/bin/httpd:ap_run_handler+0x5b 0x4
/home/trawick/inst/24-64/bin/httpd:ap_invoke_handler+0x173
/home/trawick/inst/24-64/bin/httpd:ap_process_async_reques
/home/trawick/inst/24-64/bin/httpd:0x468dc4
/home/trawick/inst/24-64/bin/httpd:0x468fb3
/home/trawick/inst/24-64/bin/httpd:ap_run_process_connecti
...
```

mod_whatkilledus report (cont.)

```
Request line (parsed):
GET :10080 /crash/
Request headers:
Host:127.0.0.1%3a10080
User-Agent:ApacheBench/2.3
Accept:*/*

Client connection:
127.0.0.1:44883->127.0.0.1:10080  (user agent at 127.0.0.1
```

mod_whatkilledus notes

- mod_whatkilledus and mod_backtrace actually work well on Windows, with great backtraces if the web server .pdb files are available.
- The original versions of mod_whatkilledus and mod_backtrace worked somewhat differently:
  - mod_backtrace and mod_whatkilledus acted independently.
  - Neither supported Windows, and mod_backtrace supported fewer Unix-y platforms.
  - mod_whatkilledus had no mechanism to filter out sensitive information.
- `http://emptyhammock.com/projects/httpd/diag/`

Comparison with httpd 2.2 — error log

- Error messages
  - No module id, pid, thread id, etc. unless the module generating the message adds it explicitly.
  - No sub-second timestamps.
  - No traceXXX levels Some messages just aren't present, because even LogLevel debug would be too noisy, or separate log files are used (mod_rewrite) which have to be managed independently.
  - No per-module LogLevel, no per-dir LogLevel (which is what allows per-client LogLevel) Custom scripting can be used to reduce the output to something readable, though nothing can be done about the volume, and that may necessitate a different scheme for rotating logs during problem determination.

Comparison with httpd 2.2 — other logs

- mod_log_debug isn't available.

Comparison with nginx 1.2.latest

A few areas to think about...

- Logging
- DTrace-ing
- Backtraces

nginx — Logging

Configure with `--debug` option so that a reasonable amount of information is available.

```
epoll add event: fd:7 op:1 ev:00000001
accept on 0.0.0.0:2080, ready: 0
posix_memalign: 000000000268C910:256 @16
*1 accept: 127.0.0.1 fd:3
*1 event timer add: 3: 60000:1352205189278
*1 epoll add event: fd:3 op:1 ev:80000001
accept on 0.0.0.0:2080, ready: 0
posix_memalign: 000000000268CA20:256 @16
*2 accept: 127.0.0.1 fd:10
*2 event timer add: 10: 60000:1352205189278
*2 epoll add event: fd:10 op:1 ev:80000001
*1 malloc: 00000000026A1FA0:1256
*1 posix_memalign: 000000000268CB30:256 @16
*1 malloc: 00000000026A2490:1024
*1 posix_memalign: 00000000026930C0:4096 @16
*1 http process request line
```

nginx — Logging

```
*1 recv: fd:3 459 of 1024
*1 http request line: "GET / HTTP/1.1"
*1 http uri: "/"
*1 http args: ""
*1 http exten: ""
*1 http process request header line
*1 http header: "Host: 127.0.0.1:2080"
*1 http header: "Connection: keep-alive"
*1 http header: "User-Agent: Mozilla/5.0 (X11; Linux x86_6
*1 http header: "Accept: text/html,application/xhtml+xml,a
*1 http header: "Accept-Encoding: gzip,deflate,sdch"
*1 http header: "Accept-Language: en-US,en;q=0.8"
*1 http header: "Accept-Charset: ISO-8859-1,utf-8;q=0.7,*;
*1 http header: "If-Modified-Since: Fri, 02 Nov 2012 21:52
*1 http header done
*1 event timer del: 3: 1352205189278
*1 rewrite phase: 0
*1 test location: "/"
```

nginx — logging for only selected clients

```
events {
    debug_connection 192.168.1.1;
    debug_connection 192.168.10.0/24;
}
```

nginx — DTrace-ing

- There's a fork of nginx 1.2.1 (slightly out of date) that includes DTrace probes (and System Tap too). (unclear how experimental this is)
- The pid provider needs to be provided the pid to trace, which is not a burden with nginx. This is the provider that allows instrumentation of arbitrary functions in the process. The nginx docs have some minimal information about using the pid provider with it.

nginx — Backtraces

- The good news is that there aren't *n* processes to look at, potentially with a number of threads in each one (depending on the httpd MPM in use).
- The not so good news is that you don't have the state of active requests/connections in such a simple representation as a function call stack.
- A backtrace for a crash should be approximately as useful as with httpd.
- For a hang or other non-crash issue, information needs to be dug out of the connection table to see what is going on, and that's not practical without a debug build.

(BTW, someone has an equivalent of mod_backtrace for nginx, but)

Recap of Jeff's toys

- Explore, collect.py
- mod_backtrace and mod_whatkilledus
- mod_hook_ar
- pgfiles.py (not mentioned; shows open files for a process group, organized to show which files are shared by different processes)

Available from

- `http://emptyhammock.com/projects/` and/or
- `http://emptyhammock.com/downloads/`

httpd materials

- httpd debugging guide, `http://httpd.apache.org/dev/debugging.html`
- `http://www.cs.virginia.edu/.../apache/apache2moddebugging.ppt`
- `http://prefetch.net/articles/debuggingapache.html`

Other topics

- http://wiki.nginx.org/Debugging
- https://forums.freebsd.org/showthread.php?p=183044
- http://www.brendangregg.com/DTrace/dtrace_oneliners.txt
- http://agentzh.org/misc/nginx/agentzh-nginx-tutorials-enuk.html