Using FastCGI with Apache HTTP Server

Jeff Trawick

November 3, 2010

Table of Contents

FastCGI

- A protocol for communicating between a web server and persistent application processes which can handle any of several different phases of requests.

FastCGI

- A protocol for communicating between a web server and persistent application processes which can handle any of several different phases of requests.
- Implemented for most web servers.

FastCGI

- A protocol for communicating between a web server and persistent application processes which can handle any of several different phases of requests.
- Implemented for most web servers.
- Implemented for most programming languages and a number of frameworks.

Fast CGI

Inputs and outputs are similar to CGI:

- environment variables
  CONTENT_LENGTH, SCRIPT_NAME, etc.

Fast CGI

Inputs and outputs are similar to CGI:

- environment variables
  CONTENT_LENGTH, SCRIPT_NAME, etc.
- input stream for request body

Fast CGI

Inputs and outputs are similar to CGI:

- environment variables
  CONTENT_LENGTH, SCRIPT_NAME, etc.
- input stream for request body
- output stream for response headers and body

Fast CGI

Inputs and outputs are similar to CGI:

- environment variables
  CONTENT_LENGTH, SCRIPT_NAME, etc.
- input stream for request body
- output stream for response headers and body
- output stream for log messages

Fast CGI

Inputs and outputs are similar to CGI:

- environment variables
  CONTENT_LENGTH, SCRIPT_NAME, etc.
- input stream for request body
- output stream for response headers and body
- output stream for log messages

But binary encoded on a stream connection or pipe (Windows).
FastCGI process waits repeatedly for new connections.

Raw CGI

```
int main(int argc, char **argv)
{
  extern char **environ;
  /* environ is {"CONTENT_LENGTH=105",
                 "SCRIPT_NAME=/path/to/foo.fcgi", etc. } */
  const char *cl_str;

  if ((cl_str = getenv("CONTENT_LENGTH")) {
    read(FILENO_STDIN,,); /* request body */
  }
  write(STDOUT_FILENO,,); /* response headers
                           * and body */
  write(STDERR_FILENO,,); /* to web server log */
}
```

Raw FastCGI

```
int main(int argc, char **argv)
{
  socket(); bind(); listen();
  while (1) {
    int cl = accept();
    read(cl, buf);
|00000 0101000100080000 0001000000000000 ........ ........|
|00010 0104000100190000 090e485454505f48 ........ ..HTTP_H|
|00020 4f53543132372e30 2e302e313a383038 OST127.0 .0.1:808|
|00030 3101040001002000 000f0f485454505f 1..... . ...HTTP_|
|00040 555345525f414745 4e54417061636865 USER_AGE NTApache|
|00050 42656e63682f322e 3301040001001000 Bench/2. 3.......|
|00060 000b03485454505f 414343455054a2f ...HTTP_ ACCEPT*/|
    write(cl, buf);
|00000 01060001041a0600 436f6e74656e742d ........ Content-|
|00010 547970653a207465 78742f706c61696e Type: te xt/plain|
|00020 0a0a787878787878 7878787878787878 ..xxxxxx xxxxxxxx|
|00030 7878787878787878 7878787878787878 xxxxxxxx xxxxxxxx|
    close(cl);
  }
```

Raw FastCGI

How the datastream is defined:

- version — 1 byte, always 1
- type — 1 byte, stuff like begin-request, abort-request, params, stdin-data, stdout-data, etc.
- request id — 2 byte request identifier
- content length — 2 byte lengh of bundled data
- padding length — 1 byte length of padding data
- the data (0–64k bytes)
- the padding (0–255 bytes)

```
|0101000100080000 0001000000000000 ........ ........|
|0104000100190000 090e485454505f48 ........ ..HTTP_H|
```

Programming support

After all, nobody would want to reinvent *that* protocol.

- FastCGI protocol libraries are available for use with Perl, Python, Ruby, C, etc., often based on the C library from Open Market.
  Code to the API to implement a FastCGI application.
  With some APIs, a properly coded FastCGI app will also work as plain CGI.
- PHP supports it transparently.
- Some frameworks support it transparently.

Perl example, moving from CGI to FastCGI

CGI-only Perl script:

```
use CGI;
my $q = CGI->new;
$mode = $q->param('mode');
print $q->header(-type => 'text/html');
print $q->start_html('hello, world'),
      $q->h1('hello, world'),
      $q->end_html;
}
```

Perl example, moving from CGI to FastCGI

CGI-only Perl script:

```
use CGI;
my $q = CGI->new;
$mode = $q->param('mode');
print $q->header(-type => 'text/html');
print $q->start_html('hello, world'),
      $q->h1('hello, world'),
      $q->end_html;
}
```

Dual-mode CGI/FastCGI Perl script:

```
use CGI::Fast;
while (my $q = CGI::Fast->new) {
  $mode = $q->param('mode');
  print $q->header(-type => 'text/html');
  print $q->start_html('hello, world'),
        $q->h1('hello, world'),
        $q->end_html;
}
```

Perl example, moving from CGI to FastCGI

CGI-only Perl script:

```
use CGI;
my $q = CGI->new;
$mode = $q->param('mode');
print $q->header(-type => 'text/html');
print $q->start_html('hello, world'),
      $q->h1('hello, world'),
      $q->end_html;
}
```

Dual-mode CGI/FastCGI Perl script:

```
use CGI::Fast;
while (my $q = CGI::Fast->new) {
  $mode = $q->param('mode');
  print $q->header(-type => 'text/html');
  print $q->start_html('hello, world'),
        $q->h1('hello, world'),
        $q->end_html;
}
```

But beware of unintentionally saved state.

Web server support for FastCGI applications

- Available for most web servers, including Apache httpd, IIS, Lighttpd, nginx, iPlanet, and others
- Typically implemented as a shared library (plug-in module) which can be loaded if the feature is desired

Who would have thought

- such an old and obvious extension to CGIs is still relevant a web lifetime later?

History

- The FastCGI protocol was originally developed in 1995–1996 for a web server from Open Market, perhaps in response to the NSAPI programming model which allowed for C language plug-ins for the Netscape (now iPlanet once again) web server.

  http://www.fastcgi.com/devkit/doc/fcgi-spec.html
- One web server implementation of particular interest, also from Open Market: mod_fastcgi 1.0 for Apache 1.0 in 1996

See the August 28, 1996 issue of Apache Week.

What happened after that?

- FastCGI was great for converting existing CGIs (often Perl) and drastically improving performance.

But:

- Native web server APIs were exploited more and more, either for existing scripting languages like Perl or new languages like PHP.
- Apache httpd modules took off. Web developers and deployers became accustomed to native code plug-ins.

What happened after that?

- FastCGI was great for converting existing CGIs (often Perl) and drastically improving performance.

But:

- Native web server APIs were exploited more and more, either for existing scripting languages like Perl or new languages like PHP.
- Apache httpd modules took off. Web developers and deployers became accustomed to native code plug-ins.

(Surplus of C programmers?)

Native module drawbacks

- Overall resource use often larger when app runs in the web server, especially for prefork model
  - memory
  - connections to database, LDAP, etc.

  Resources are often left behind on any thread/process that occasionally runs the application — underutilized.

Native module drawbacks

- Overall resource use often larger when app runs in the web server, especially for prefork model
  - memory
  - connections to database, LDAP, etc.

  Resources are often left behind on any thread/process that occasionally runs the application — underutilized.
- Introduces instability into server

Native module drawbacks

- Overall resource use often larger when app runs in the web server, especially for prefork model
  - memory
  - connections to database, LDAP, etc.

  Resources are often left behind on any thread/process that occasionally runs the application — underutilized.
- Introduces instability into server
- Collisions between requirements of different modules

Native module drawbacks

- Overall resource use often larger when app runs in the web server, especially for prefork model
  - memory
  - connections to database, LDAP, etc.

  Resources are often left behind on any thread/process that occasionally runs the application — underutilized.
- Introduces instability into server
- Collisions between requirements of different modules
- Generally unable to support multiple script interpreter versions

Native module drawbacks

- Overall resource use often larger when app runs in the web server, especially for prefork model
  - memory
  - connections to database, LDAP, etc.

  Resources are often left behind on any thread/process that occasionally runs the application — underutilized.
- Introduces instability into server
- Collisions between requirements of different modules
- Generally unable to support multiple script interpreter versions
- Potential lack of thread safety, or expensive locking

But with FastCGI

- Often the required application thread/process count is a fraction of that of the web server (so resources not left behind on threads/processes occasionally used).

But with FastCGI

- Often the required application thread/process count is a fraction of that of the web server (so resources not left behind on threads/processes occasionally used).
- A particular application usually can't introduce instability into the server, so basic services and other applications are unaffected.

But with FastCGI

- Often the required application thread/process count is a fraction of that of the web server (so resources not left behind on threads/processes occasionally used).
- A particular application usually can't introduce instability into the server, so basic services and other applications are unaffected.
- Different applications can use different libraries, interpreter versions, framework versions, etc.

But with FastCGI

- Often the required application thread/process count is a fraction of that of the web server (so resources not left behind on threads/processes occasionally used).
- A particular application usually can't introduce instability into the server, so basic services and other applications are unaffected.
- Different applications can use different libraries, interpreter versions, framework versions, etc.
- Independent start/stop of web server and application

But with FastCGI

- Often the required application thread/process count is a fraction of that of the web server (so resources not left behind on threads/processes occasionally used).
- A particular application usually can't introduce instability into the server, so basic services and other applications are unaffected.
- Different applications can use different libraries, interpreter versions, framework versions, etc.
- Independent start/stop of web server and application
- Independent identity or chroot env vs. web server and other applications

Criticisms of FastCGI

- protocol is too complex to implement

  but a number of language bindings share the same core, the Open Market C-language interface

Criticisms of FastCGI

- protocol is too complex to implement

  but a number of language bindings share the same core, the Open Market C-language interface

- defines many more features than are actually used (or at least *working* in practice)
  - lack of support for FastCGI filtering
  - lack of support for FastCGI management queries
  - AAA broken in mod_fcgid for years (still waiting for 2.3.6)

  who needs it?

Criticisms of FastCGI

- protocol is too complex to implement

  but a number of language bindings share the same core, the Open Market C-language interface

- defines many more features than are actually used (or at least *working* in practice)
  - lack of support for FastCGI filtering
  - lack of support for FastCGI management queries
  - AAA broken in mod_fcgid for years (still waiting for 2.3.6)

  who needs it?

- generally troublesome implementations

  just switch from mod_fastcgi to mod_fcgid or httpd to Lighttpd or ...

Concerns for some special situations

- Protocol is complex enough that unsanitized input could expose bugs in non-mainstream protocol support *in app but also in server*

Concerns for some special situations

- Protocol is complex enough that unsanitized input could expose bugs in non-mainstream protocol support *in app but also in server*
- Care needed with TCP to protect access
  - Instant auth: Just set REMOTE_USER
  - Throw garbage at the TCP port, see what happens

AF_UNIX has filesystem permissions and is system-only.

Competitors

- HTTP — If you use HTTP you can interoperate with almost anything. *But if you use HTTP you have a lot to implement to be able to interoperate with what people will throw at you.*
- SCGI — http://python.ca/scgi/protocol.txt
  - about 100 lines, so easy to implement yourself if existing library support isn't available or suitable
  - commonly used FastCGI capabilities, plus sendfile

    One unfortunate omission: doesn't provide a way for routing stderr messages (integration into web server logs).

Competitors

- CGI — why not, if load isn't an issue?
- AJP — not just for Java application servers
- custom

These have varying infrastructure to help with process management and protocol.

(And of course mod_*foo* and JVM or CLR-based interpreters.)

FastCGI modules for Apache httpd

Two contenders in this space

- mod_fastcgi
    - http://www.fastcgi.com/
    - the original
    - often referred to as outdated and unmaintained, though it is still actively used and there was a snapshot created in 2009 (last release was 2.4.6 in 2007)
    - supports httpd 1.3–2.2 (and probably needs just a few tweaks to support 2.4-dev)

FastCGI modules for Apache httpd

Two contenders in this space

- mod_fastcgi
  - http://www.fastcgi.com/
  - the original
  - often referred to as outdated and unmaintained, though it is still actively used and there was a snapshot created in 2009 (last release was 2.4.6 in 2007)
  - supports httpd 1.3–2.2 (and probably needs just a few tweaks to support 2.4-dev)
- mod_fcgid
  - http://httpd.apache.org/mod_fcgid/
  - originally created by Ryan Pan in 2004 and published under GPL; hosted at SourceForge
  - donated and relicensed to ASF in 2009; now developed and released as a subproject of Apache httpd
  - supports httpd 2.0–2.4-dev

Proxy scheme modules

Two separately maintained proxy scheme modules:

- mod_proxy_fcgi in Apache httpd 2.4-dev
  covered more later
- mod_proxy_fcgi at SourceForge
  Because it is limited to httpd 2.0, it doesn't support load balancing.
  not discussed further

They support only externally managed servers over TCP/IP.

Which mod_fcgid?

- 2.2 was the only release for a long time, and some important patches accumulated in the source tree, SourceForge bug db, and in private trees.
- Not all of that has made its way to ASF mod_fcgid.

Capability differences

mod_fastcgi has

- load balancing to FastCGI processes
- statically defined and externally managed FastCGI processes
- support for FastCGI apps which can handle multiple concurrent requests or include their own process management
- other less-important features

Capability differences

mod_fcgid has

- better option configuration for dynamically started FastCGI processes
- monitoring of processes via mod_status reports
- *arguably* better configuration of process management parameters
- *arguably* better documentation

neither has X-Sendfile, unlike Lighttpd

Logging differences

- mod_fastcgi writes a lot of information about process management at LogLevel warn
  2009 snapshot has same bogus error info logged that has been around for a long time
- mod_fcgid uses LogLevel info for similar information
  but before 2.3.5 it used LogLevel notice — unsuppressable – for some common messages

Other possible considerations — Licensing

- mod_fastcgi is not considered a free software license. mod_fastcgi and any derivatives are licensed for the sole purpose of implementing FastCGI specs defined or endorsed by Open Market.
  (The Open Market license for FastCGI libraries has no such restriction.)
- No issues with ASL 2.0.

Other possible considerations — Dev processes

- mod_fcgid — Code repos, releases, security issues, bug reporting, development discussions, etc. are the same as the server itself.
- mod_fastcgi — Code on a git repo at http://repo.or.cz, not much recent history of releases or security process, bug reporting on the fastcgi.com-hosted mailing list.

Comments on the web

- mod_php faster

Comments on the web

- mod_php faster
- PHP over FastCGI about as fast

Comments on the web

- mod_php faster
- PHP over FastCGI about as fast
- mod_fastcgi faster than mod_fcgid

Comments on the web

- mod_php faster
- PHP over FastCGI about as fast
- mod_fastcgi faster than mod_fcgid
- mod_fcgid faster than mod_fastcgi

Comments on the web

- mod_php faster
- PHP over FastCGI about as fast
- mod_fastcgi faster than mod_fcgid
- mod_fcgid faster than mod_fastcgi
- mod_fcgid more stable than mod_fastcgi

Comments on the web

- mod_php faster
- PHP over FastCGI about as fast
- mod_fastcgi faster than mod_fcgid
- mod_fcgid faster than mod_fastcgi
- mod_fcgid more stable than mod_fastcgi
- mod_fcgid results in less system memory consumed than mod_fastcgi

Comments on the web

- mod_php faster
- PHP over FastCGI about as fast
- mod_fastcgi faster than mod_fcgid
- mod_fcgid faster than mod_fastcgi
- mod_fcgid more stable than mod_fastcgi
- mod_fcgid results in less system memory consumed than mod_fastcgi
- mod_fastcgi has issues with dead FastCGI processes

Comments on the web

- mod_php faster
- PHP over FastCGI about as fast
- mod_fastcgi faster than mod_fcgid
- mod_fcgid faster than mod_fastcgi
- mod_fcgid more stable than mod_fastcgi
- mod_fcgid results in less system memory consumed than mod_fastcgi
- mod_fastcgi has issues with dead FastCGI processes
- AF_UNIX sockets not reliable

Comments on the web

- mod_php faster
- PHP over FastCGI about as fast
- mod_fastcgi faster than mod_fcgid
- mod_fcgid faster than mod_fastcgi
- mod_fcgid more stable than mod_fastcgi
- mod_fcgid results in less system memory consumed than mod_fastcgi
- mod_fastcgi has issues with dead FastCGI processes
- AF_UNIX sockets not reliable
- PHP before version x.y.z had bad process management

Comments on the web

- mod_php faster
- PHP over FastCGI about as fast
- mod_fastcgi faster than mod_fcgid
- mod_fcgid faster than mod_fastcgi
- mod_fcgid more stable than mod_fastcgi
- mod_fcgid results in less system memory consumed than mod_fastcgi
- mod_fastcgi has issues with dead FastCGI processes
- AF_UNIX sockets not reliable
- PHP before version x.y.z had bad process management
- "Paul said MoinMoin didn't work with fcgid."

Comments on the web

What is true?

- Everything?
- Nothing?
- 10%, the rest just being bad configuration and/or simple but undiagnosed bugs?

The definitive source of knowledge

Let’s put this to rest once and for all!

The definitive source of knowledge

Let's put this to rest once and for all!

- "mod_fastcgi sucks"

The definitive source of knowledge

Let's put this to rest once and for all!

- "mod_fastcgi sucks"

  3 hits on google

The definitive source of knowledge

Let’s put this to rest once and for all!

- “mod_fastcgi sucks”

  3 hits on google

- “mod_fcgid sucks”

The definitive source of knowledge

Let's put this to rest once and for all!

- "mod_fastcgi sucks"

  3 hits on google

- "mod_fcgid sucks"

  0 hits on google

Quick and dirty performance test, no configuration

1k response, 1 concurrent client (ab -n 10000 -c 1), ignore first run

Handler	Requests/second
fastcgi	1073, 1281, 1495, 1065, 1145
fcgid	969, 1102, 965, 990, 1077
cgid	35, 35

Quick and dirty performance test, no configuration

8k response, 120 concurrent clients (ab -n 10000 -c 120), ignore first run

Handler	Requests/second	Processes
fastcgi	2680, 2662, 2646, 2646, 2647	1
fcgid	2933, 2906, 2826, 2958, 2944	15
cgid	62, 61	*n*

mod_fcgid: syscalls for script when already started

```
[pid 31334] semop(5636097, {{0, -1, SEM_UNDO}}, 1) = 0
[pid 31334] semop(5636097, {{0, 1, SEM_UNDO}}, 1) = 0
[pid 31334] socket(PF_FILE, SOCK_STREAM, 0) = 16
[pid 31334] connect(16, {sa_family=AF_FILE,
    path="/home/trawick/ApacheCon/inst/logs/fcgidsock/31221.2"}, 110) = 0
[pid 31334] setsockopt(16, SOL_TCP, TCP_NODELAY, [1], 4)
    = -1 EOPNOTSUPP (Operation not supported)
[pid 31334] fcntl64(16, F_GETFL)        = 0x2 (flags O_RDWR)
[pid 31334] fcntl64(16, F_SETFL, O_RDWR|O_NONBLOCK) = 0
[pid 31334] gettimeofday({1288725238, 241008}, NULL) = 0
[pid 31334] writev(16, [{"\1\1\0\1\0\10\0\0", 8}, {"\0\1\0\0\0\0\0\0", 8},
    {"\1\4\0\1\2\314\0\0", 8},
    {"\t\16HTTP_HOST127.0.0.1:8080\17\17HTTP_"..., 716}, {"\1\4\0\1\0\0\0\0", 8},
    {"\1\5\0\1\0\0\0\0", 8}, {NULL, 0}], 7) = 756
[pid 31334] read(16, 0x90341a8, 8192)   = -1 EAGAIN (Resource temporarily unavailable)
[pid 31334] poll([{fd=16, events=POLLIN}], 1, 40000) = 1 ([{fd=16, revents=POLLIN}])
[pid 31334] read(16, "\1\6\0\1\4\32\6\0Content-Type: text/plain"..., 8192) = 1088
[pid 31334] gettimeofday({1288725238, 241663}, NULL) = 0
[pid 31334] close(16)                   = 0
[pid 31334] semop(5636097, {{0, -1, SEM_UNDO}}, 1) = 0
[pid 31334] semop(5636097, {{0, 1, SEM_UNDO}}, 1) = 0
```

mod_fastcgi: syscalls for script when already started

```
[pid 31305] stat64("/home/trawick/ApacheCon/inst/logs/fastcgi/dynamic/
    5cb5cbf10efa67823cdd3e00cbac2a8d", {st_mode=S_IFSOCK|0600, st_size=0, ...}) = 0
[pid 31305] socket(PF_FILE, SOCK_STREAM, 0) = 16
[pid 31305] gettimeofday({1288726660, 499152}, NULL) = 0
[pid 31305] connect(16, {sa_family=AF_FILE, path="/home/trawick/ApacheCon/inst/logs/
    fastcgi/dynamic/5cb5cbf10efa67823cdd3e00cbac2a8d"}, 84) = 0
[pid 31305] fcntl64(16, F_GETFL)        = 0x2 (flags O_RDWR)
[pid 31305] fcntl64(16, F_SETFL, O_RDWR|O_NONBLOCK) = 0
[pid 31305] gettimeofday({1288726660, 502717}, NULL) = 0
[pid 31305] select(17, [16], [16], NULL, {3, 96435}) = 1 (out [16], left {3, 96423})
[pid 31305] write(16, "\1\1\0\1\0\10\0\0\0\1\0\0\0\0\0\0\1\4
    \0\1\0\31\0\0\t\16HTTP_H"..., 886) = 886
[pid 31305] gettimeofday({1288726660, 503126}, NULL) = 0
[pid 31305] select(17, [16], [], NULL, {3, 96026}) = 1 (in [16], left {3, 95540})
[pid 31305] gettimeofday({1288726660, 503821}, NULL) = 0
[pid 31305] read(16, "\1\6\0\1\4\32\6\0Content-Type: text/plain"..., 8192) = 1088
[pid 31305] fcntl64(16, F_GETFL)        = 0x802 (flags O_RDWR|O_NONBLOCK)
[pid 31305] fcntl64(16, F_SETFL, O_RDWR) = 0
[pid 31305] setsockopt(16, SOL_SOCKET, SO_LINGER, {onoff=0, linger=0}, 8) = 0
[pid 31305] close(16)
```

Syscall summary

mod_fcgid has, mod_fastcgi doesn't:

- setsockopt(TCP_NODELAY) *EOPNOTSUPP*
- writev() for 756 bytes *with wasted element*
- read() *EAGAIN*
- poll()
- two lock/unlock sequences

Syscall summary

mod_fastcgi has, mod_fcgid doesn't:

- stat()
- two more calls to gettimeofday()
- write() for 886 bytes
- select()
  bad for platforms with small FD_SETSIZE and heavily threaded MPM child processes
- fcntl(turn off non-blocking)
- setsockopt(disable linger)

The unexpected socket calls make more sense when using TCP sockets.

Simple CGI and FastCGI configuration

Start with cgi:

```
Alias /fastcgi/ \
/home/trawick/ApacheCon/inst/fastcgi/

<Location /fastcgi>
Options +ExecCGI
SetHandler cgi-script
</Location>
```

IOW, enable the ExecCGI option and set the handler appropriately. (ScriptAlias kind-of does that)
Change the handler name to fcgid-script (mod_fcgid) or fastcgi-script (mod_fastcgi).

More classic CGI configuration

```
<Location /myapp/>
  AddHandler prettify txt
  Action prettify /tools/prettify.pl
</Location>

<Directory /www/tools/>
  Options +ExecCGI
  SetHandler cgi-script
</Directory>
```

Again, change the handler name to fcgid-script (mod_fcgid) or fastcgi-script (mod_fastcgi).

Is that all?

It might be, unless...

- default timeouts or other I/O settings aren't okay
  connect timeout, read/write timeout, hang detection

Is that all?

It might be, unless...

- default timeouts or other I/O settings aren't okay
  connect timeout, read/write timeout, hang detection
- default process management isn't okay

Is that all?

It might be, unless...

- default timeouts or other I/O settings aren't okay
  connect timeout, read/write timeout, hang detection
- default process management isn't okay
  limits on numbers of processes, rules for shrinking the pool

Is that all?

It might be, unless...

- default timeouts or other I/O settings aren't okay
  connect timeout, read/write timeout, hang detection
- default process management isn't okay
  limits on numbers of processes, rules for shrinking the pool
- minor protocol adjustments, environment variables, etc.

Is that all?

It might be, unless...

- default timeouts or other I/O settings aren't okay
  connect timeout, read/write timeout, hang detection
- default process management isn't okay
  limits on numbers of processes, rules for shrinking the pool
- minor protocol adjustments, environment variables, etc.

Additional configuration is likely except for sites with only relatively simple FastCGI applications.

mod_fcgid

All mod_fcgid configuration from this point!

Hung process detection

By default, if a request does not *complete* within five minutes the application will be terminated.
No way to disable. *Fixme.*

```
# my report generates output over a long period of
# time; don't kill it
FcgidBusyTimeout 3600

# kill anything that doesn't respond within 30 second
FcgidBusyTimeout 30
```

I/O timeouts (hung process?)

By default, if no data can be read or written within 40 seconds, the application will be terminated.

```
# my report generates output over a long period of
# time; don't kill it
FcgidBusyTimeout 3600
# oh, and there are long pauses between generation
# of any output
FcgidIOTimeout  300
```

Process management

Simple stuff:

- FcgidMaxProcesses — global limit on number of processes
- FcgidMaxProcessesPerClass — limit on number of processes per application
- FcgidIdleTimeout — termination after idle for this long
- FcgidMaxRequestsPerProcess — termination after handling this many requests
- FcgidProcessLifetime — termination after alive for this long

Tuning of process management algorithms

Spawn score: internal calculation which represents process activity for a FastCGI application; used to determine if a new instance (process) can be created.

```
# Set this high.  If actual score is higher for an app,
# more instances can't be created.
FcgidSpawnScoreUpLimit      7000
```

Tuning of process management algorithms

Spawn score: internal calculation which represents process activity for a FastCGI application; used to determine if a new instance (process) can be created.

```
FcgidSpawnScoreUpLimit      7000

# Default value.  Each process creation adds this to the
# score.
FcgidSpawnScore                  1
```

Tuning of process management algorithms

Spawn score: internal calculation which represents process activity for a FastCGI application; used to determine if a new instance (process) can be created.

```
FcgidSpawnScoreUpLimit      7000
FcgidSpawnScore                1

# By default, termination increases the score.  But why?
# If a process goes away, create additional headroom for
# creating a replacement.
FcgidTerminationScore         -1
```

Tuning of process management algorithms

Spawn score: internal calculation which represents process activity for a FastCGI application; used to determine if a new instance (process) can be created.

```
FcgidSpawnScoreUpLimit      7000
FcgidSpawnScore                1
FcgidTerminationScore         -1

# Subtracted from the score each second.
FcgidTimeScore                 3
```

Ugly tuning of process management algorithms

mod_fcgid scans for certain conditions at configurable intervals. The default values for the intervals are quite high for some — 120 seconds.

```
# Scan for processes which have exceeded idle timeout every
# second.
FcgidIdleScanInterval          0
```

Ugly tuning of process management algorithms

mod_fcgid scans for certain conditions at configurable intervals. The default values for the intervals are quite high for some — 120 seconds.

```
FcgidIdleScanInterval          0

# Scan for processes which need to be terminated every second.
FcgidErrorScanInterval         0
```

Ugly tuning of process management algorithms

mod_fcgid scans for certain conditions at configurable intervals. The default values for the intervals are quite high for some — 120 seconds.

```
FcgidIdleScanInterval         0
FcgidErrorScanInterval        0

# Scan for zombie processes every second.
#   (Why don’t we just call waitpid() to see if any children
#   exited?)
FcgidZombieScanInterval       0
```

Wrappers

- a command which will run for certain requests — by container or extension
- typically is outside of the web space
- typically is a script which encapsulates command-line parameters and environment settings such as envvars and ulimits

```
<Location /phpapp/>
  AddHandler fcgid-script .php
  Options +ExecCGI
  FcgidWrapper /usr/local/bin/php-wrapper .php
</Location>
```

Command-specific options

Most directives apply to every app in a certain container (vhost or per-dir).
FcgidCmdOptions allows many directives to be applied for a single specific command.

```
FcgidCmdOptions /path/to/info.pl \
    IdleTimeout 30 \
    InitialEnv VHOST=www2.example.com \
    IOTimeout 5 \
    MaxRequestsPerProcess 10000
```

Jeff, this is where you show the sample server-status page.

PHP and FastCGI

Not at all uncommon...

FastCGI required or recommended for PHP with

- nginx
- Lighttpd
- Zeus
- IIS

PHP and FastCGI

Not at all uncommon...

FastCGI required or recommended for PHP with

- nginx
- Lighttpd
- Zeus
- IIS

Can work fine with Apache httpd too

Special considerations

- PHP FastCGI processes normally exit after 500 requests

  Syncronize mod_fcgid and PHP limits to avoid 500 error[1].

[1] The HTTP error code and the request count are both 500. Conspiracy?

Special considerations

- PHP FastCGI processes normally exit after 500 requests

  Syncronize mod_fcgid and PHP limits to avoid 500 error[1].

  In PHP wrapper:

  ```
  PHP_FCGI_MAX_REQUESTS=10000
  ```

  In fcgid configuration:

  ```
  FcgidMaxRequestsPerProcess 10000
  ```

  or just set PHP_FCGI_MAX_REQUESTS to 0 to disable

[1]The HTTP error code and the request count are both 500. Conspiracy?

Special considerations

- PHP FastCGI process management ineffective (wasted) with mod_fcgid, which routes only single concurrent requests to the socket of a process which it has spawned,

  Leave PHP child process management disabled (PHP_FCGI_CHILDREN=0).

Special considerations

- PHP FastCGI process management ineffective (wasted) with mod_fcgid, which routes only single concurrent requests to the socket of a process which it has spawned,

  Leave PHP child process management disabled (PHP_FCGI_CHILDREN=0).

  not an issue with mod_fastcgi, though there are lingering complaints that PHP process management is problematic anyway

Special considerations

But:

- With PHP process management, single cache can be used concurrently by many processes.

Special considerations

But:

- With PHP process management, single cache can be used concurrently by many processes.
- Without PHP child process management, PHP opcode caches are not very effective. Cache is serially reused within single process when the same fcgid-spawned process handles another request.

More oddities

- PHP flags in .htaccess files — no longer respected when you move from mod_php to FastCGI
- on Windows, mod_php strips the drive letter from SCRIPT_NAME; mod_fcgid doesn't

The future

NEAR future

mod_fcgid 2.3.6 expected to be rolled today and available *soon*

- fix some remaining AAA issues
- gracefully handle application returning no output
- fix problems with some mass-vhost modules
- allow apps more time to exit at httpd shutdown
- fix some bogus defaults
- fixes for other, more obscure, situations

Apache httpd 2.4

mod_proxy_fcgi

- FastCGI apps externally managed (spawn-fcgi or httpd's new, less–featureful fcgistarter)
- Needs more TLC before it is ready for use

mod_proxy_fcgi

```
# disable backend keepalive connection
ProxyPass /myapp/ fcgi://localhost:4000/ disablereuse=on
```

- Start the app via spawn-fcgi or the new fcgistarter or similar.
- For a single threaded app, the starter can open *n* instances on the same port.
- Hope they don't die, or implement your own process monitoring/management.

MEDIUM future

- Route to apps which can handle multiple concurrent requests

MEDIUM future

- Route to apps which can handle multiple concurrent requests
- Get mod_proxy_fcgi working

MEDIUM future

- Route to apps which can handle multiple concurrent requests
- Get mod_proxy_fcgi working
- Resolve other major bugs/complants

MEDIUM future

- Route to apps which can handle multiple concurrent requests
- Get mod_proxy_fcgi working
- Resolve other major bugs/complants
- Profile to find areas needing optimization

MEDIUM—LONG future

Revisit areas of duplication between mod_fcgid and the rest of the server.

- process management — duplicated among fcgid, rewrite, cgi*, fcgidstarter, etc.
- request and response buffering — duplicated among fcgid, buffer, possibly others
- general application model — duplicated among fcgid and some hypotetical mod_scgi

Thank you!

- mod_fcgid user support: See http://httpd.apache.org/userslist.html
- mod_fcgid development: dev@httpd.apache.org