Choosing a Web Architecture for Perl

Perrin Harkins
We Also Walk Dogs

This used to be an easy decision

- Netscape, or Apache for you hippies
- CGI, or NSAPI if you enjoy core dumps

Server-side development matured, and it stayed easy

- Apache became the standard web server
- mod_perl became the way to run Perl
- FastCGI was looking peaky
    - Outside of Amazon's reality distortion field anyway

Then two things happened

- Ruby came along
  - Renewed interest in FastCGI
  - Thanks, Ruby!
- Non-blocking I/O became fashionable
  - Memcached
  - Lighttpd
  - Nginx

Now we have many choices

- Web servers
  - Apache
  - Lighttpd
  - Nginx
  - Squid
  - Perlbal
  - Varnish
- Protocols
  - FastCGI
  - HTTP
  - SCGI

FastCGI

- Many implementations for different servers
- Daemons are managed by the web server or an external process
- Has hooks for auth, logging, and filtering, but not always implemented and rarely used

HTTP

- Reverse proxy forwards HTTP requests to apache/mod_perl backend

SCGI

- ”Simple CGI”
- Stealing good ideas from Python
- Simpler than FastCGI, so theoretically could be faster

The graveyard of alternative Perl daemons

- SpeedyCGI
- PersistentPerl
- Both replace the #! line and daemonize your program

Biases

- Long-time mod_perl contributor
- Member of Apache Software Foundation

Don't believe a word I say!

Apache 2.2 mod_proxy + mod_perl

- Frontend apache runs threaded worker MPM
- Backend apache runs prefork MPM
- All popular apache modules (mod_ssl, mod_rewrite, mod_security, mod_deflate, caching, auth) are available
- mod_proxy_balancer provides load balancing

Apache 2.2 mod_fastcgi

- Frontend runs threaded worker MPM
- As with mod_proxy, all popular apache modules are available
- Three poorly-named configurations:
  - ”Static” starts a set number of procs at server startup
  - ”Dynamic” starts a new proc every time it gets a request up to a configured limit
  - ”Standalone” sends requests to daemon you start

Apache 2.2 mod_fastcgi

- Can run separate versions of same package name in separate daemons
  - True for mod_perl too if you run multiple backends, but less obvious
- Seamless restart for code upgrades in standalone mode
  - Start a new set of procs on same socket and shut down the old ones
  - Could get close with two mod_perl backends, but not simple

Lighttpd mod_fastcgi

- Non-blocking I/O server
- Supports static and standalone FastCGI
- Supports Authorizer role
- Solid selection of modules with equivalents of most popular apache ones
- Same restart advantages as FastCGI on Apache
- Load balancing across FastCGI daemons

Lighttpd mod_scgi

- Same Lighttpd, different protocol
- Sketchy docs

nginx FastCGI

- Another non-blocking I/O server
- Supports standalone FastCGI only
- Does not support Authorizer role
- Good module selection
- Same restart advantages

nginx proxy + mod_perl

- Load balancing

What about running naked mod_perl?

- Static requests
- Buffering for slow clients
- Keep-Alive

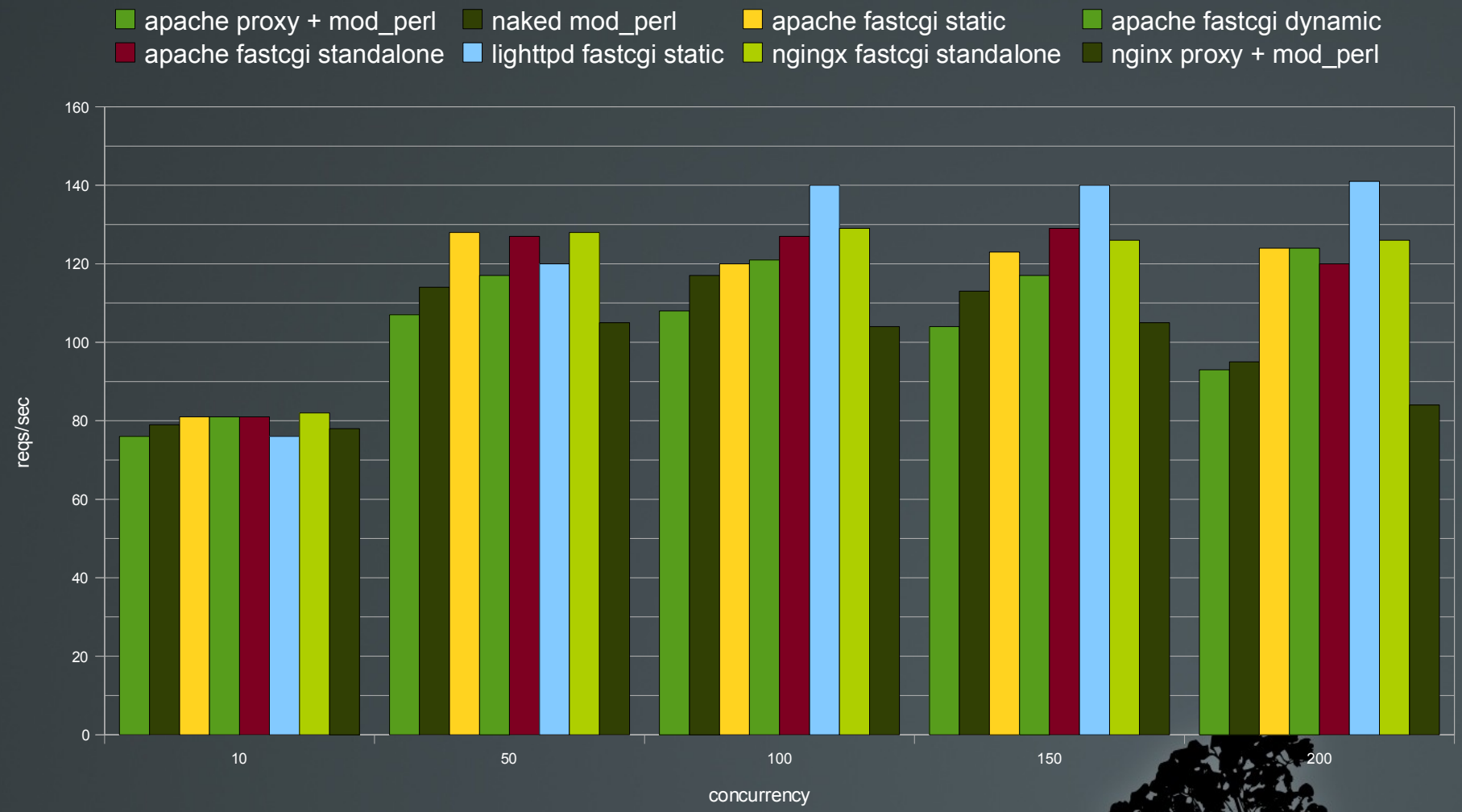
Benchmark methodology

- Very simple Catalyst app
  - Sleeps for 0.1 seconds
  - Returns 35K of HTML (perl.com homepage)
- Brief run of ab (Apache Bench) with intended concurrency to warm up server
- ab tests at concurrency of 10, 50, 150, 200

General findings

- FastCGI static mode is difficult to use with apache/mod_fastcgi
  - Trial and error to find magic number
  - Worked better with Lighttpd
- FastCGI standalone mode is unusable on Lighttpd
  - Load balancing feature marks daemons as down if they're too busy to answer right away and rejects requests
- SCGI Perl implementations incomplete

Benchmark results



Not really apples to apples

- mod_perl does other things
- Filters on non-perl content (PHP, etc.)
- Customize server and request phases
- Replace HTTP with your own protocol
    - Apache 2 + mod_perl 2 is a server kit

Thanks!

And please help me add more servers...