Perlbal: Reverse Proxy & Webserver

- Brad Fitzpatrick
  - brad@danga.com
- Danga Interactive
  - Open Source company
  - Services
    - LiveJournal.com
    - PicPix.com (soon) / pics.livejournal.com
  - Tools
    - DBI::Role
    - memcached
    - mogilefs (distributed filesystem)
    - Perlbal

Features

- Reverse proxy load balancer
- Buffers backend responses
  - frees heavy mod_perl/mod_php/mod_foo early
- Can internally “reproxy” to another URL or file
- Web server mode (after thought)
  - survives when thttpd hasn't, uses epoll
- 'epoll', for handling many connections efficiently. 'sendfile' for less CPU usage
- Console/HTTP management

Why not just _____?

- Buffering
  - BIG-IP does ~16k
    - configurable in 9.x, but limited memory on a single box
  - Apache does ~128k (tcp send buffer). can tell kernel to increase.
- “Internal proxy”
  - client sees no redirect
  - switch between internal servers
  - auth/URI trans in mod_perl/etc
  - quick webserver (thttpd/TUX/perlbal)
    - skipping ahead: IO::Sendfile!
- Custom LB / no proxy connect errors

Web server mode

- Why reproxy to another webserver?
- Why not reproxy to a file?
- Hard work already done
- nonblock network easy
- nonblock VFS stuff harder
- nonblock stat() / open() hard
  - once open, sendfile() to nonblock socket no prob: IO::SendFile

Our setup

- Two BIG-IPs (active / standby)
  - vip livejournal.com:80 = 4 perlbals
- Previously:
  - to 4 mod_proxy, to mod_rewrite, to external rewrite map “prg:” doing sendstats (like mod_backhand)
- Sendstats no longer necessary:
  - persistent backend connections
    - 1.0 only, don't have to dechunk, dynamic responses rare
  - verify backends with OPTIONS
    - talking to Apache, not kernel

Perlbal::Socket

- Singleton event loop
- constructor registers self with event loop; single threaded, non-blocking, event-based
- POE-like
- uses epoll (Linux 2.6) or poll
  - IO::Poll flakey. use IO::Poll::_poll which is reliable.
  - epoll via perl's 'syscall'. IO::Epoll was flakey.
- Later found useful for
  - ddlockd, mogilefsd, etc
  - pushed down into Danga::Socket

use fields

- so damn cool
- compile-time member checking
  - 'use strict' for OO
  - when typed (my Foo $a), then hash access are actually array accesses ($a->{bar} compiles to $a->[18])
- run-time member checking
  - when not typed
  - hash+array lookup. still strict.
- confidence to do big OO projects

Linux::AIO

- uses linux's clone()
- shared pipe to cloned child to alert of waiting jobs
- fd parent can [e]poll on. Linux::AIO calls closure when told to (via fd readiness)
- before this, cycles
- not portable
  - may change to thttpd style: unix socket to blocking IO workers. pass fds.

doo dads

- management port
  - every console command can be in config file or set at runtime
  - console commands that look like HTTP are treated as such, and web UI runs (ssh -L8065: proxy:8065... http://localhost:8065/)
- “connect-ahead”

Perlbal Plug-ins

- hooks into core Perlbal code
  - as needed now. documented. could add more.
- currently plug-ins for
  - stats (counts connections)
  - queues (shows queue depths)
  - palette modifications on GIF/PNG
    - lets user customize their colors and makes accompanying images match theme, without dynamically generating the images.
    - palette table is in first few bytes, then sendfile() the rest.
  - de-animate GIFs (upcoming, byte toggle)

MogileFS distributed file system; HTTP PUT support

- storage nodes have disks
- files are on 'n' disks on different hosts
- Min replica count 'n' based on class of file
- Dozens of NFS problems
  - nfs_inode_cache, corruption, export limitations
- Perlbal!
  - storage nodes run Perlbal for GETs
  - how to PUT?
  - PUT support, optional (off by default)
  - change wrapper script so no config needed

Code: Linux::AIO

```
Linux::AIO::aio_stat($file, sub {
    return if $self->{closed};  # client away
    return $self->_simple_response(404) unless -e _;
    ....
    Linux::AIO::aio_open($file, 0, 0, sub {
        my $rp_fd = shift;
        ....
        $self->state('xfer_disk');
        $self->tcp_cork(1);  # cork writes to self
        $self->write($res->to_string_ref);
        $self->reproxy_fd($rp_fd, $size);
    });
});
```

Code: epoll wrappers

```
our $HAVE_SYSCALL_PH = eval { require 'syscall.ph'; 1 };
our $SYS_epoll_create = eval { &SYS_epoll_create };
our $SYS_epoll_ctl = eval { &SYS_epoll_ctl };

# ARGS: (size)
sub epoll_create {
    my $epfd = eval { syscall($SYS_epoll_create, $_[0]) };
    return -1 if $@;
    return $epfd;
}

# ARGS: (epfd, op, fd, events)
sub epoll_ctl {
  syscall($SYS_epoll_ctl, $_[0]+0, $_[1]+0, $_[2]+0,
          pack("LLL", $_[3], $_[2]));
}
# <snip> epoll_wait...
```

Code: use fields

```
package Rect;
use base Shape;
use fields qw(width height);
sub new {
   my Rect $self = shift;
   $self = fields::new($self) unless ref $self;
   ($self->{width}, $self->{height}) = @_;
   return $self;
}
...
my Rect $rect = Rect->new(10, 20);
print “$rect->{hieght}\n”;

#              ^^^^^^
#    COMPILE-TIME ERROR!  (or runtime error without typing)
# Compiles to:
   $rect->[14]    (or whatever index)
```

Questions?

the end.

Brad Fitzpatrick
brad@danga.com

Mark Smith
marksmith@danga.com

Danga Interactive
http://www.danga.com/

All our stuff's Open Source:
http://cvs.danga.com/
http://cvs.livejournal.org/