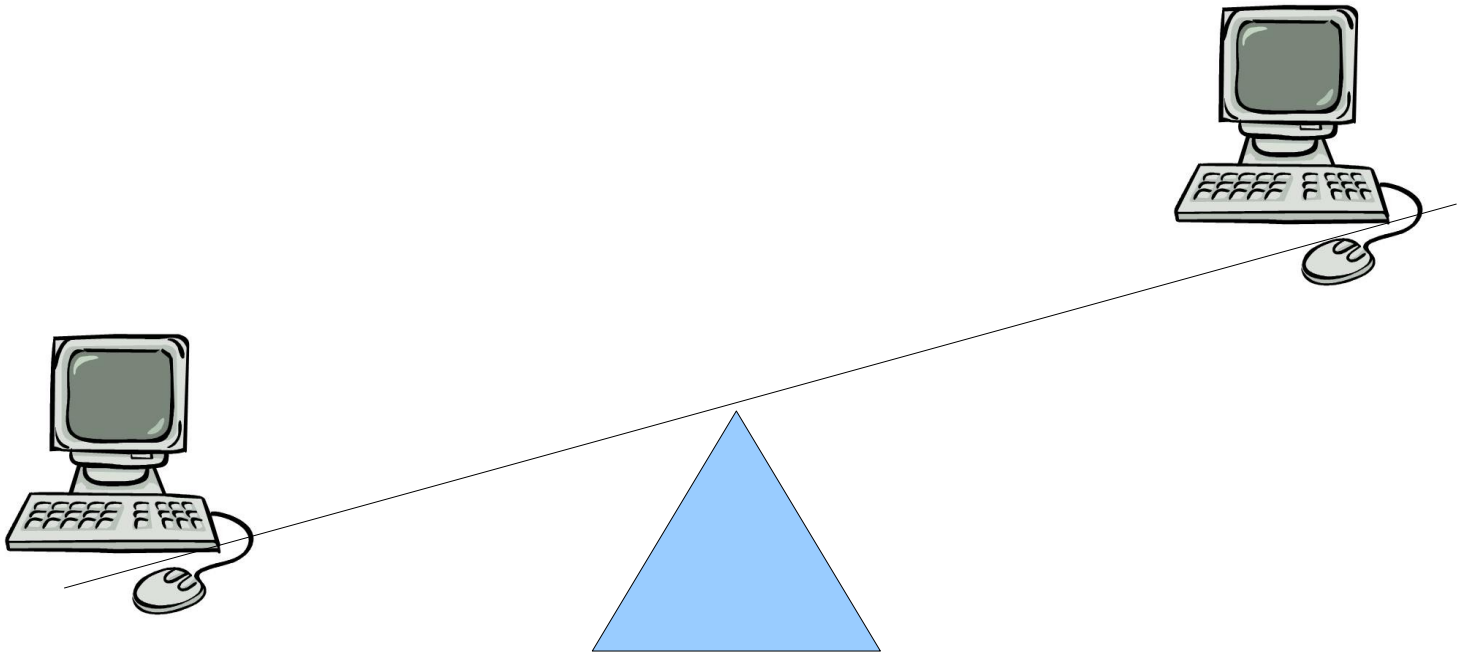
# Load Balancing with Perlbal

Katherine Spice

MiltonKeynes.pm

15/10/09

# What is Load Balancing?



# Why and when might you use Load Balancing?

- Capacity: you aren't able to meet the demand on your service with a single server
- Resilience: you want to be able to cope with the failure of a server

# Considerations in choosing a Load Balancer

## Features

- Type of traffic
  - Layer 3 / 4 (IP/TCP) vs Layer 7 (application specific, e.g. HTTP)
- Balancing algorithm
  - random choice
  - round robin
  - manual weighting
  - automated weighting based on environment such as:
    - reported load
    - response time
    - available connections

## Cost

- Commercial Hardware:

  F5 BigIP, Citrix NetScaler, Cisco ACE

  Easy, fast, ££££ to £££,£££!

- Commercial Software:

  Citrix VPX, Zeus ZXTM

  up to ££,£££

- OSS:

  LVS, Balance, Perlbal

  Free

So, if you're after a free, high performance, software HTTP load balancer.....

Perlbal

# Background

First release was in 2005

Danga Interactive / Brad Fitzpatrick
http://www.danga.com/perlbal/

*“needed internal redirects, tried to hack it onto 4 others load balancers, but too painful. Easier to write our own, then once we had it we were able to start doing tons more tricks and had a lot more visibility into what was going on. Also our performance went up a ton, as Perlbal's load balancing is just better than anything else we tried. (and we went through a ton of load balancers)”*

# Perlbal

Perlbal is a Perl-based reverse proxy load balancer and web server.

It processes hundreds of millions of requests a day just for LiveJournal, Vox and TypePad and dozens of other "Web 2.0" applications.

# Features

- Maintains pool of connected backend connections to reduce turnover
- Intelligent load balancing based on what backend connections are free for a new request.
- Almost everything can be configured or reconfigured on the fly without needing to restart the software
- Has a high priority queue for sending requests through to backends quickly (based on cookies or URI or host).
- Configurable header management before sending request to backend
- Internal redirection to file or URL(s) transparent to the client.
- Can verify that a backend connection is talking to a webserver and not just the kernel's listen queue before sending client requests at it.
- Is extendable via plugins

# Obtaining and Installing

Current version: 1.73

CPAN
http://search.cpan.org/dist/Perlbal/

```
perl -MCPAN -e shell
cpan-> install Perlbal
```

Ubuntu packages
https://edge.launchpad.net/~awmcclain/+archiv

# Configuring

Default config file: /etc/perlbal/perlbal.conf

```
CREATE POOL my_apaches
  POOL my_apaches ADD 10.0.0.10:8080
  POOL my_apaches ADD 10.0.0.11:8080
  POOL my_apaches ADD 10.0.0.12
  POOL my_apaches ADD 10.0.0.13:8081

CREATE SERVICE balancer
  SET listen          = 0.0.0.0:80
  SET role            = reverse_proxy
  SET pool            = my_apaches
  SET persist_client  = on
  SET persist_backend = on
  SET verify_backend  = on
ENABLE balance
```

# Configuring alt.

```
CREATE POOL dynamic
  SET nodefile = conf/nodelist.dat

CREATE SERVICE balancer2
  SET listen           = 0.0.0.0:81
  SET role             = reverse_proxy
  SET pool             = dynamic
ENABLE balancer2
```

# Running

```
$ ./perlbal --help

Usage: perlbal [OPTS]
  --help            This usage info
  --version         Print perlbal release version
  --config=[file]   Specify Perlbal config file
                    (default: /etc/perlbal/perlbal.conf)
  --daemon          Daemonize



$ ./perlbal -d
```

# Managing

Add this to config file:

```
# always good to keep an internal management port open:
CREATE SERVICE mgmt
  SET role   = management
  SET listen = 127.0.0.1:60000
ENABLE mgmt

$ telnet 127.0.0.1:60000
pool my_apaches ADD 10.0.0.14

SHOW POOL

CREATE POOL new_apaches
pool new_apaches add 10.0.0.15
set balancer pool = new_apaches
```

# Monitoring

Perlbal provides the following via the management interface, all in machine readable format.

* CPU usage (user, system)
* Total requests served across all services
* Requests service by individual backends
* Perlbal uptime
* All connected sockets
* Outstanding connections to backends
* Backends that have recently failed verification
* Pending backend connections by service
* Total of all socket states by socket type
* Size (in seconds and number of connections) of all queues
* State of reproxy engine (queued requests, outstanding requests, backends)
* Loaded plugins per service

# Extending

Perlbal supports the concept of having per-service and global plugins that can override many parts of request handling and behavior, via hooks.

## Global hooks

```
Perlbal::register_global_hook('foo', sub { return 0; });
```

## Service handler hooks

```
$service->register_hook('bar', sub {
    # do something
    return 1;
});
```

## Service general hooks

# Hooks

Examples:

HANDLER start_web_request Perlbal::ClientHTTP
When a 'web' service has gotten headers and is about to serve it... return a true value to cancel the default handling of web requests.

HANDLER start_send_file Perlbal::ClientHTTPBase
Called when we've opened a file and are about to start sending it to the user using sendfile. Return a true value to cancel the default sending.

HANDLER start_serve_request Perlbal::ClientHTTPBase, $uri_ref
Called when we're about to serve a local file, before we've done any work. You can change the file served by modifying $uri_ref, and cancel the process by returning a true value.

# Plugin Config

In perlbal.conf

```
LOAD MyPlugin

CREATE SERVICE balancer
  SET listen           = 0.0.0.0:80
  SET role             = reverse_proxy
  SET pool             = my_apaches
  SET persist_client   = on
  SET persist_backend  = on
  SET verify_backend   = on
  SET plugins           = MyPlugin
ENABLE balancer
```

# Plugin Structure

```
package Perlbal::Plugin::MyPlugin;

use strict;
use warnings;

# Called when we are loaded, do set up and global commands
  here
sub load { return 1; }

# Clear our global commands
sub unload { return 1; }

# called when we're being added to a service
sub register { return 1; }

# called when we're no longer active on a service
sub unregister { return 1; }

1;
```

# Extracted from Perlbal::Plugin::Log at http://www.eamondaly.com/perl/Perlbal/Log.pm

```
sub load {
    my $class = shift;

    Perlbal::register_global_hook('manage_command.logname', sub {
        my $mc = shift->parse(qr/^logname\s+(?:(\w+)\s+)?\s*=\s*(.+)\s*$/,
                              "usage: LOGNAME [<service>] = <file>");

        my ($svc_name, $logname) = $mc->args;

        unless ($svc_name ||= $mc->{ctx}{last_created}) {
            return $mc->err("omitted service name not implied from context");
        }

        my $ss = Perlbal->service($svc_name);
        return $mc->err("Service '$svc_name' is not a web_server service")
            unless $ss && $ss->{role} eq "web_server";

         my $fh;

         $logname =~ s/^\"//;
         $logname =~ s/\"$//;

         if ($logname =~ s/^\|\s*//) {
             $fh = new IO::Pipe;
             eval { $fh->writer($logname) };

             # Note that the ability to capture any errors here will
             # depend largely on your OS and shell
             return $mc->err("Failed to open '$logname': $@")
               if $@;
         }
         else {
             $fh = new IO::File $logname, O_WRONLY|O_APPEND|O_CREAT;
             return $mc->err("Failed to open '$logname': $!")
               unless $fh;
         }

         $fh->autoflush(1);

         $logobjs{$svc_name}->{'service'} = $ss;
         $logobjs{$svc_name}->{'fh'} = $fh;

         return $mc->ok;
    });

    return 1;
}
```

```
sub register {
    my ($class, $svc) = @_;

    $svc->register_hook('Log', 'start_web_request', sub {
     my Perlbal::ClientHTTP $client = shift;
     $client->{'scratch'}->{'Log:start_web_request'} =
  [ gettimeofday() ];

     return 0;
    });

    $svc->register_hook('Log', 'end_web_request', sub {
     &log_request(shift);

        return 0;
    });

    return 1;
}
```

```
sub log_request {
    my Perlbal::ClientHTTP $client = shift;
    my Perlbal::HTTPHeaders $req_headers = $client->{req_headers};
    my Perlbal::HTTPHeaders $res_headers = $client->{res_headers};

    my $start_time = delete $client->{'scratch'}->{'Log:start_web_request'};
    my $req_time = sprintf('%.3f', &tv_interval($start_time)) if $start_time;

    if ($client && $req_headers && $res_headers) {
my $svc = $client->{'service'};

if (my $fh = $logobjs{$svc->{'name'}}->{'fh'}) {
    my $auth;

    if ($req_headers->{'headers'}{'authorization'} =~ /^Basic (.+)/) {
($auth, undef) = split(/:/, decode_base64($1), 2)
    }

    my $referer = $req_headers->{'headers'}{'referer'};
    $referer =~ s/"/\\"/g;

    my $ua = $req_headers->{'headers'}{'user-agent'};
    $ua =~ s/"/\\"/g;

    printf $fh qq{%s - %s [%s] "%s" %d %d "%s" "%s" %s %s %s %s\n},
      $client->peer_ip_string,
      $auth || '-',
      strftime('%d/%b/%Y:%H:%M:%S %z', localtime),
      $req_headers->{'requestLine'},
      $res_headers->response_code,
      $res_headers->content_length,
      $referer || '-',
      $ua || '-',
      $req_headers->{'headers'}{'host'} || '-',
      $req_time || '-',
      $svc->name,
      $svc->role;
}
    }
}
```

```
# If there's no existing hook for end_web_request in
# http_response_sent, add one.
if (! exists &Perlbal::Service::end_web_request) {
    # Add dummy hook
    *Perlbal::Service::end_web_request = sub { };

    # Copy the original http_response_sent
    *_http_response_sent_orig =
  *Perlbal::ClientHTTPBase::http_response_sent;

    # Override with our own, which includes the hook
    *Perlbal::ClientHTTPBase::http_response_sent =
  *_http_response_sent;
}

sub _http_response_sent {
    my Perlbal::ClientHTTPBase $self = shift;

    $self->{'service'}->run_hook('end_web_request', $self);

    _http_response_sent_orig($self);
}
```

## Documentation

"*Much more documentation needs to happen...*"

http://www.danga.com/perlbal/

Active user group at:
http://groups.google.com/group/perlbal

## Releases

Perlbal-1.60 - 24 Oct 2007
Perlbal-1.70 - 09 Mar 2008
Perlbal-1.71 - 14 Sep 2008
Perlbal-1.72 - 22 Sep 2008

**Latest version:** Perlbal-1.73 - 05 Oct 2009

And from the release announcement
"*We're aiming at doing a second release in 1-3 weeks...*"

Thank you and any questions?