Concurrent Requests and measuring performance with ApacheBench

PSGI | Starman | plackup | ApacheBench

Prev Next

When you build a web application at first there is only one user: you. Everything works fine. When you launch the web site, sooner or later more people visit it, and there might be even several people at the same time.

What happens then? How can your server handle concurrent requests?

Echo with Plackup

Let's see a simple PSGI script that would echo back whatever we type in the text-box.

```
#!/usr/bin/perl
use strict;
use warnings;

use Plack::Request;

my $app = sub {
    my $env = shift;

    my $html = get_html();

    my $request = Plack::Request->new($env);

    if ($request->param('field')) {
        $html .= 'You said: ' . $request->param('field');
    }

    return [
        '200',
        [ 'Content-Type' => 'text/html' ],
        [ $html ],
    ];
};

sub get_html {
```

```
        return q{
          <form>

          <input name="field">
          <input type="submit" value="Echo">
          </form>
          <hr>
        }
    }
```

Save the above code as `echo.psgi` and run it as `plackup echo.psgi`.

We can now browse to `http://127.0.0.1:5000/` where we'll see the input box. If we type in "hello" and press enter, it will take us to `http://127.0.0.1:5000/?field=hello` and display `You said: hello`. It works. Let's use ab - ApacheBench to measure the performance.

We run `ab http://127.0.0.1:5000/?field=hello` and this is the output:

```
This is ApacheBench, Version 2.3 <$Revision: 1554214 $>
Copyright 1996 Adam Twiss, Zeus Technology Ltd, http://www.zeustech.net/
Licensed to The Apache Software Foundation, http://www.apache.org/

Benchmarking 127.0.0.1 (be patient).....done


Server Software:        HTTP::Server::PSGI
Server Hostname:        127.0.0.1
Server Port:            5000

Document Path:          /?field=hello
Document Length:        127 bytes

Concurrency Level:      1
Time taken for tests:   0.001 seconds
Complete requests:      1
Failed requests:        0
Total transferred:      257 bytes
HTML transferred:       127 bytes
Requests per second:    734.75 [#/sec] (mean)
Time per request:       1.361 [ms] (mean)
Time per request:       1.361 [ms] (mean, across all concurrent requests)
Transfer rate:          184.41 [Kbytes/sec] received

Connection Times (ms)
```

```
              min  mean[+/-sd] median   max
Connect:        0    0   0.0      0       0
Processing:     1    1   0.0      1       1
Waiting:        1    1   0.0      1       1
Total:          1    1   0.0      1       1
```

The interesting bits are:

But what happens if the actual request takes longer. For example 2 seconds?

We change the code a bit to sleep for 2 seconds before echoing back the string we typed in.

```
1.          $html .= 'Start: ' . time . '<br>';
            sleep 2;
            $html .= 'You said: ' . $request->param('field') . '<br>';
            $html .= 'End: ' . time . '<br>';
```

Stop (Ctrl-C) and start the server again.

The `Time taken for tests` went up to 2.005 seconds.

What if there are several clients at the same time?

ab, the ApacheBench can have many parameters. `-n` will let us tell it how many requests to send in total and `-c` will configure the level of concurrency: How many requests to send at the same time.

Let's run `ab -n 3 -c 3 http://127.0.0.1:5000/?field=hello` that is, 3 request at the same time. The result is disappointing: `Time taken for tests: 6.010 seconds`. While we asked to run all 3 of them in parallel, they still took 3*2 = 6 seconds.

That's because `plackup` is a very basic server that can only handle one request at a time. If I ran it again, this time a total of 10 request at 10 concurrency level, it would take 10*2=20 seconds.

Let's try Starman which is a "High-performance preforking PSGI/Plack web server".

We stop the server using Ctrl-C and start it again using `starman echo.psgi`

If we run `ab -n 3 -c 3 ...` it will report 2 seconds total time. Starman has several processes (workers) and each one can handle a request separately.

If we run `ab -n 10 -c 10 ...`, the total elapsed time is 4 seconds. That's because by default Starman starts with 5 workers. So every 5 requests can be done in parallel. (If we run -n 11 -c 11 it will take a total of 6 seconds.)

We can also stop the Starman server and run it again, this time with 20 workers: `starman --workers 20 ...`

Now it can handle as many as 20 concurrent requests but it uses a lot more memory.