Load test your web applications part 1 - Apache ab

Author : Tobias Hofmann

Date : June 4, 2012

Note: 1st published on SCN on 4.6.2012

With ab

Load testing an application is more than a task to prepare for the go-live: it has to cover all phases of the lifecycle. It allows tracing the behavior of the application over time, allowing identifying when performance degeneration happened. It's part of a toolset to avoid "yesterday it worked fine …".

From the many load testing tools the ones for testing the end user perspective are the ones most useful. For applications that have a web interface this means to test the load from the browser perspective. This way, not only a specific part of the application like the DB is tested: all components involved to generate the HTML output are tested: the web server, application server, network connection, backend applications, DB, and so on. There are several load testing tools available, some are endorsed by SAP and are made available by SAP partners. They have one thing in common (beside the price): they are not easy to use. Because of this they normally get added at the quality test phase, and that is already too late. I will focus here on tools that can be freely downloaded, installed, are simply to set up and use and have a community so finding solutions on the internet is not a big deal.

The Apache web server comes with a simple tool: ab (http://httpd.apache.org/docs/2.0/programs/ab.html). Basically, that tool does everything you need to test the performance of a web page. You can use ab to upload a file, set cookies to find out HTTP server limitations (like 413), emulate authentication, and so on. The intended use case of ab is to test the Apache web server and the impact of configuration parameters on its performance, but as it simulates quite well the load of several browsers and a flexible number of requests, it can be used to test every web page. You can try out parameter changes and see how they impact the performance, how your code handles several requests, find out the impact of the network, and many more.

To run a test that includes 1000 requests, distributed over 10 threads to a URL, the syntax is:

```
\apache\bin>ab -n 1000 -c 10 -w http://scn.sap.com/welcome
```

Output:

```
Server Software:        Apache-Coyote/1.1
Server Hostname:        scn.sap.com
Server Port:            80

Document Path:          /welcome
Document Length:        137320 bytes

Concurrency Level:      10
Time taken for tests:   211.899 seconds
Complete requests:      1000
Failed requests:        0
Write errors:           0
Total transferred:      138069286 bytes
HTML transferred:       137320000 bytes
Requests per second:    4.72 [#/sec] (mean)
Time per request:       2118.991 [ms] (mean)
Time per request:       211.899 [ms] (mean, across all concurrent requests)
Transfer rate:          636.31 [Kbytes/sec] received

Connection Times (ms)
              min  mean[+/-sd] median   max
Connect:       17   29  15.9     24     270
Processing:  1035 2082 2046.8   1577   22987
Waiting:      866 1795 2047.4   1283   22762
Total:       1078 2110 2046.1   1607   23008

Percentage of the requests served within a certain time (ms)
  50%   1607
  66%   1795
  75%   1924
  80%   2004
  90%   2634
  95%   4991
  98%   9289
  99%  12898
 100%  23008 (longest request)
```

```
<p>
 This is ApacheBench, Version 2.3 <i>&lt;$Revision: 655654 $&gt;</i><br>
 Copyright 1996 Adam Twiss, Zeus Technology Ltd, http://www.zeustech.net/<br>
 Licensed to The Apache Software Foundation, http://www.apache.org/<br>
</p>
<p>
Completed 100 requests
Completed 200 requests
Completed 300 requests
Completed 400 requests
Completed 500 requests
Completed 600 requests
```

Ab gives already some important information like requests/sec, number of failed requests and transfer rate. It's an easy to use tool but pretty low level. Running ab against a local installation of SAP Portal:

```
Server Software:        SAP
Server Hostname:        ivml2005
Server Port:            50000

Document Path:          /index.html
Document Length:        505 bytes

Concurrency Level:      10
Time taken for tests:   1.739 seconds
Complete requests:      1000
Failed requests:        0
Write errors:           0
Total transferred:      770000 bytes
HTML transferred:       505000 bytes
Requests per second:    575.01 [#/sec] (mean)
Time per request:       17.391 [ms] (mean)
Time per request:       1.739 [ms] (mean, across all concurrent requests)
Transfer rate:          432.38 [Kbytes/sec] received

Connection Times (ms)
              min  mean[+/-sd] median   max
Connect:        0    1   1.0      1      14
Processing:     4   16   5.8     15      47
Waiting:        3   13   4.9     13      40
Total:          5   17   5.8     16      48

Percentage of the requests served within a certain time (ms)
  50%     16
  66%     19
  75%     21
  80%     22
  90%     24
  95%     28
  98%     34
  99%     36
 100%     48 (longest request)
```

This example makes also clear that one important aspect of load testing is the network connection. Instead of making the requests over Wifi to the URL of the 1st example somewhere in the internet and thus slowing down the number of requests, here the requests are done locally and I get an impressive 575 requests per seconds. The key takeaway here is: do your load testing not only at one location, do it at several locations inside your network to rule out or find network related bottlenecks: start near your server for optimizing the performance of your application and then move slowly away. Of course, you should define for each of the locations a threshold.

To test your single application, you need

1. A direct link
2. User authentication (if implemented)

The direct link is easy to get. The user authentication with ab is tricky, but the –C parameter does the trick. This will not give real world results as ab will do the tests as the same user, but it gives an overview of the response time of the application. The application is reading some data from the request object and some KM properties.

```
Server Software:        SAP
Server Hostname:        ivml2005
Server Port:            50000

Document Path:          /irj/servlet/prt/portal/prtroot/Training.Test1
Document Length:        1390 bytes

Concurrency Level:      10
Time taken for tests:   0.924 seconds
Complete requests:      100
Failed requests:        0
Write errors:           0
Total transferred:      170400 bytes
HTML transferred:       139000 bytes
Requests per second:    108.22 [#/sec] (mean)
Time per request:       92.405 [ms] (mean)
Time per request:       9.241 [ms] (mean, across all concurrent requests)
Transfer rate:          180.08 [Kbytes/sec] received

Connection Times (ms)
              min  mean[+/-sd] median   max
Connect:        0    2   3.3      1      20
Processing:    23   81 108.1     53     744
Waiting:       18   75 108.0     47     740
Total:         25   84 108.0     56     747

Percentage of the requests served within a certain time (ms)
  50%     56
  66%     62
  75%     71
  80%     81
  90%    137
  95%    210
  98%    609
  99%    747
 100%    747 (longest request)
```

I ran the test several time and what you cannot see here is that the number of request went up from 49 to 92 and then stayed there. That's an easy application that only reads some request parameters. Where serving a simple html page came back with 575 hits/sec, the application running inside the portal is not – as expected - as fast.

Now I can add and remove some code to find out if the application runs faster or slower and this way find out where the performance bottleneck is.