1.2.3 Stress Testing

The system described throughout the scheduler’s deliverables is a system which, by the very nature of its usage, will experience varying degrees of client traffic: There will be varying volume of users login in to the system during the day versus the middle of the night or the dates of class registration versus other days between or in the middle of any given semester. This volume of users will tend to be condensed during the same higher traffic times (i.e. during the day of course registration) and more spread out during other times. These periods of heavy load on the system can be simulated and exaggerated beyond normal operation the system with a script efficiency tool. The resulting stress test can be applied to a system in order to determine the robustness, response and availability of a correct behavior for any system that may experience variable operation. The generic test is a black box test that can be applied to any system algorithm. However, the particular programming language and methods are a good indication

For the purpose of this deliverable, Apache Benchmark (ab) along with a PHP code in order console in order to repeat multiple htpwas used as a dynamic software verification and validation method in order to produce repeatable testing with quantitative request times (connect, processing, waiting, and total), transferred bytes (total, HTML, and document) for a given number of requests to be performed and the number of requests that occur simultaneously. A greater number of requests may represent simply a longer period of time (a week vs an hour). However, if a greater number of these requests occur at the same time (concurrency) than what is normal, then we are implementing a stress test in which we can observe several of the stress related defects.

The dynamic apachebenchmark test the PHP code and the server by attributing a number –n of requests and a value –c for the amount of simultaneous requests to be performed on every component of the server. This test however simulates a single device sending requests; in order to simulate many more devices, each with their own number set of requests, a PHP code was added to the apachebenchmark tool in order to multitask a number –r of url request repetition performed by each client and their value –c for the list of simultaneous clients.

Thus we have –c concurrent clients requesting the url’s –r times each making each their –n requests with –c multiples.

This test is much more likely to lead to a failure of the code and an incident once that code runs causing an incident that may or may not be handled because of the larger number of simultaneous and total requests.

The PHP classes are: ezab and abrunner

class eZAB

{

static $version = '0.3-dev';

static $defaults = array(

// 'real' options

/// How much troubleshooting info to print. <, 3 and above prints response codes (404, 200, etc.), 2 and above prints warnings and info."

/// Real life testing seem to tell a different story though...

'verbosity' => 1, // -v verbosity

'children' => 2, // -c concurrency Number of multiple requests to make

'tries' => 10, // -n requests Number of requests to perform

'timeout' => 0, // -t timelimit Seconds to max. wait for responses

'auth' => false,

'proxy' => false,

'proxyauth' => false,

'target' => '',

'keepalive' => false,

'head' => false,

'interface' => '',

'respencoding' => false,

'httpversion' => CURL\_HTTP\_VERSION\_NONE,

'cookies' => array(),

'skippercentiles' => false,

'extraheaders' => array(),

// 'internal' options

'childnr' => false,

'parentid' => false,

// the actual script path (self)

'self' => \_\_FILE\_\_,

'php' => 'php',

'outputformat' => 'text',

'haltonerrors' => true,

'command' => 'runparent' // allowed: 'helpmsg', 'versionmsg', 'runparent', 'runchild'

);

class ABRunner

{

static $version = '0.1-dev';

static $defaults = array(

// 'real' options

'label' => '',

'server' => 'http://localhost', // Server hostname (the prefix for urls below).

'urls' => 'index.php', // List of urls to test. Use double quotes around, separate them with spaces

'urlsfile' => '',

'repetitions' => 1000, // The number of times each client requests each url

'concurrencies' => '100 10', // List of concurrent clients to use

'dognuplot' => false,

'doaggregategraph' => false,

'ab' => 'ab',

'summary\_file' => 'summary.txt',

'output\_dir' => 'test\_logs',

'sleep' => 1,

// 'internal' options

'verbosity' => 4,

'self' => \_\_FILE\_\_,

'outputformat' => 'text',

'haltonerrors' => true,

'command' => 'runtests',

'abopts' => array()

);

Failure to meet response time requirements.

The System cannot be evaluated without completing a quantitative process to measure the response. This shall follow with more explanation.

Failure to run using particular configurations of hardware, operating systems and external libraries.

-libraries, and proper server are a critical issue in the installation and proper functioning of the Laravel framework. The absence of key libraries, PHP 5.4, PHP composer, Apache and MySQL within a proper server database are critical to the functioning of the system. The system will simply not exhibit the functionality given by the library –without warning.

Failure to gracefully handle resource shortage.

The Laravel framework implemented with a Heroku engine has the ability allow the database manager to scale in and out as well as automatic scaling to accommodate a varying number of clients in order to avoid downtime.

Failure to make resources available when they are no longer required.

A failure in the responsiveness of the SQL database can result in a cascade of failed requests and may result in a failure of the apache software and reboot incident –this is due to No code limiting the frequency of requests by any one client once a failed request has been observed.

Failure to fully recover from its own failure state or that of a related system.

And inappropriate SQL query can result into a failure of apache’s ability to respond to incoming requests which could overload incident of the system.

A failure in the PHP Laravel or composer plugins can result in a failure to interpret the code other than a string.

Testing with ApacheBenchmark:

Single instances of a client requesting a specific number –n of times, of which –c are concurrent.

In the following tables, there are 4 different instances of these simple stress tests:

First the concurrency level was changed:

|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Server Hostname | | | | Document Length (bytes) | | | Server Software | | | |
| schedule-heroku.herokuapp.com | | | | 2 036 | | | Apache | | | |
| Concurrency level | | Time for tests  (ms) | | | | Complete Requests | | | Failed Requests | |
| 100 | | 15 807 | | | | 500 | | | 0 | |
| Total transferred  (bytes) | | | | HTML transferred (bytes) | | | Transfer rate  (Kbytes/sec) | | | |
| 1 104 000 | | | | 1 018 000 | | | 68.21 | | | |
| Connection Times  (ms) | | | | | | | | | | |
|  | min | | | | mean | | median | | | max |
| Connect | 22 | | | | 31 | | 30 | | | 815 |
| Processing | 60 | | | | 2663 | | 2937 | | | 3010 |
| waiting | 54 | | | | 1479 | | 1473 | | | 2999 |
| total | 92 | | | | 2694 | | 2967 | | | 3784 |
| Time to a percent of completion (ms) | | | | | | | | | | |
| 50 | | | 75 | | | | | 90 | | |
| 2 967 | | | 3 008 | | | | | 3 034 | | |

|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Server Hostname | | | | Document Length (bytes) | | | Server Software | | | |
| schedule-heroku.herokuapp.com | | | | 2 036 | | | Apache | | | |
| Concurrency level | | Time for tests  (ms) | | | | Complete Requests | | | Failed Requests | |
| 500 | | 17 995 | | | | 500 | | | 0 | |
| Total transferred  (bytes) | | | | HTML transferred (bytes) | | | Transfer rate  (Kbytes/sec) | | | |
| 1 104 000 | | | | 1 018 000 | | | 59.91 | | | |
| Connection Times (ms) | | | | | | | | | | |
|  | min | | | | mean | | median | | | max |
| Connect | 24 | | | | 36 | | 30 | | | 3030 |
| Processing | 50 | | | | 10 382 | | 10 412 | | | 17 918 |
| waiting | 50 | | | | 10 374 | | 10 404 | | | 17 915 |
| total | 80 | | | | 10 418 | | 10 444 | | | 17 949 |
| Time to a percent of completion (ms) | | | | | | | | | | |
| 50 | | | 75 | | | | | 90 | | |
| 10 444 | | | 14 218 | | | | | 16 511 | | |

And Secondly the Complete requests was changed more drastically:

|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Server Hostname | | | | Document Length (bytes) | | | Server Software | | | |
| schedule-heroku.herokuapp.com | | | | 2 036 | | | Apache | | | |
| Concurrency level | | Time for tests  (ms) | | | | Complete Requests | | | Failed Requests | |
| 20 000 | | 303 427 | | | | 100 000 | | | 0 | |
| Total transferred  (bytes) | | | | HTML transferred (bytes) | | | Transfer rate  (Kbytes/sec) | | | |
| 220 800 000 | | | | 203 600 000 | | | 68.06 | | | |
| Connection Times  (ms) | | | | | | | | | | |
|  | min | | | | mean | | median | | | max |
| Connect | 20 | | | | 32 | | 30 | | | 3 036 |
| Processing | 4 202 | | | | 570 449 | | 626 140 | | | 663 901 |
| waiting | 1 319 | | | | 317 062 | | 317 014 | | | 651 050 |
| total | 4 234 | | | | 570 481 | | 626 170 | | | 663 931 |
| Time to a percent of completion (ms) | | | | | | | | | | |
| 50 | | | 75 | | | | | 90 | | |
| 626 170 | | | 635 745 | | | | | 657 250 | | |

Reduced request by a factor of 10.

|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Server Hostname | | | | Document Length (bytes) | | | Server Software | | | |
| schedule-heroku.herokuapp.com | | | | 2 036 | | | Apache | | | |
| Concurrency level | | Time for tests  (ms) | | | | Complete Requests | | | Failed Requests | |
| 10 000 | | 313 216 | | | | 10 000 | | | 0 | |
| Total transferred  (bytes) | | | | HTML transferred (bytes) | | | Transfer rate  (Kbytes/sec) | | | |
| 2 204 000 | | | | 2 036 000 | | | 68.84 | | | |
| Connection Times  (ms) | | | | | | | | | | |
|  | min | | | | mean | | median | | | max |
| Connect | 20 | | | | 31 | | 30 | | | 3 032 |
| Processing | 1 1142 | | | | 159 379 | | 161 661 | | | 312 704 |
| waiting | 355 | | | | 158 748 | | 161 003 | | | 312 347 |
| total | 1 174 | | | | 159 410 | | 161 689 | | | 312 738 |
| Time to a percent of completion (ms) | | | | | | | | | | |
| 50 | | | 75 | | | | | 90 | | |
| 161 689 | | | 237 593 | | | | | 312 738 | | |

These tables only represent one instance of a client requestings –they would be variable depending on the server current capacity and load. These tests give very little indication as to how the system could be pushed past it’s limits.

In order to attempt this true stress testing, the PHP classes that multiply the number of imaginary clients and their requests were used to get and average responsiveness. The variables, -n, -r and –c were set in order to recreate 1000 students 10 or 1 at a time, each with 100 requests that are made 1 at a time.

It Should be noted that the server host was changed to http;//localhost:8000 –which reduced the connect times to virtually 0ms.

--------------------------------------------------------------------------------------------------------------------------

### Start Time: Wed, 06 Apr 2016 10:55:38 +0200

Testing http://localhost:8000/index.php, concurrency: 100, iterations: 100000

Command: ab -n 100000 -c 100 "http://localhost:8000/index.php"

This is ABRunner, Version 0.1-dev

### Start Time: Wed, 06 Apr 2016 12:36:22 +0200

Testing http://localhost:8000/index.php, concurrency: 100, iterations: 100000

Command: ab -n 100000 -c 100 "http://localhost:8000/index.php"

This is ABRunner, Version 0.1-dev

### Start Time: Wed, 06 Apr 2016 12:37:45 +0200

Testing http://localhost:8000/index.php, concurrency: 100, iterations: 100000

Command: ab -n 100000 -c 100 "http://localhost:8000/index.php"

Testing http://localhost:8000/index.php, concurrency: 10, iterations: 10000

Command: ab -n 10000 -c 10 "http://localhost:8000/index.php"

This is ABRunner, Version 0.1-dev

### Start Time: Wed, 06 Apr 2016 12:44:17 +0200

Testing http://localhost:8000/index.php, concurrency: 100, iterations: 100000

Command: ab -n 100000 -c 100 "http://localhost:8000/index.php"

This is ABRunner, Version 0.1-dev

### Start Time: Wed, 06 Apr 2016 12:47:04 +0200

Testing http://localhost:8000/index.php, concurrency: 1, iterations: 1000

Command: ab -n 1000 -c 1 "http://localhost:8000/index.php"

Testing http://localhost:8000/index.php, concurrency: 10, iterations: 10000

### End Time: Wed, 06 Apr 2016 13:02:12 +0200

---------------------------------------------------------------------

This is a total time of 2: 07: 26 for 3.2 Million requests: an average Of 419 requests every second. For the purpose of this large test the more detailed time keeping was turned off in order to save resources.

A shorter test was then used with no concurrency –this results in fewer requests per second, however a much smaller time to do each request.

---------------------------------------------------------------------------------------------------------------------------------

### Start Time: Wed, 06 Apr 2016 13:22:37 +0200

Testing http://localhost:8000/index.php, concurrency: 1, iterations: 100

Command: ab -n 100 -c 1 "http://localhost:8000/index.php"

Requests per second: 12.03 [#/sec] (mean)

Time per request: 83.128 [ms] (mean)

Failed requests: 0

Testing http://localhost:8000/index.php, concurrency: 10, iterations: 1000

Command: ab -n 1000 -c 10 "http://localhost:8000/index.php"

### End Time: Wed, 06 Apr 2016 13:24:13 +0200

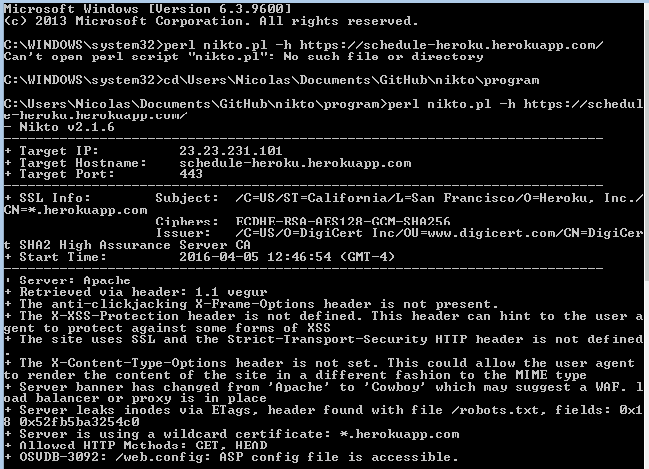
---------------------------------------------------------------------------------------------------------------------------------

These tests give a brief view into the resource done by the server in order to give the best responsiveness in each situation.

1.2.4 Security Testing

SQL and HTML injection can be used to verify the security of a system and its capacity to leak important system information about the structure of the object oriented HTML/PHP or of the database schema; such as the exact HTML/PHP structure, Files, Extensions, Updates or even Database structure and contents. This is clearly an outcome to be avoided: thus a Laravel PHP framework was implemented in order to prevent eventual security breaches –especially by injection. The higher level abstraction of the frameworks helps to achieve this.

If this nikto is directly implemented into the server location then many information is readily available:



However, if the system is properly uploaded and the nikto is ran from outside the system, then no web server is detected at all.