KIWIRE 1.5 ENTERPRISE PERFORMANCE BENCHMARK

Reviewed By:

Company	Name	Department	Date
Synchroweb	Pang Wee Tak	I.T	5/9/2014

Related Documents

Document Name	Application and Release Number	Revision Number	Dated

Contents

1. Purpose of This Document

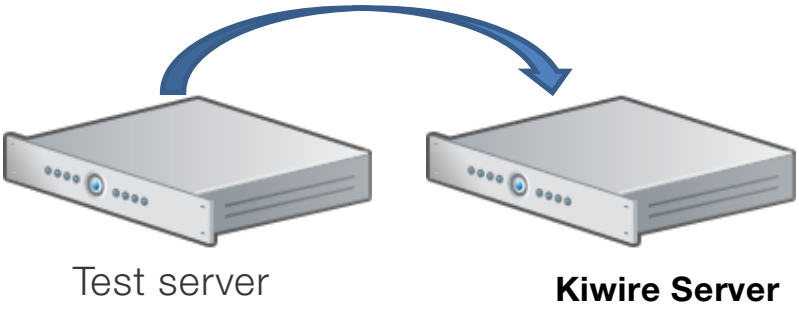
This document intent is to showcase a validated benchmark report of load stress test of kiwire 1.5 software

2. Scope

The tests will be conducted base on the standard version on a standard off the shelf hardware

3. Hardware

Hardware chosen are standard hardware from Dell , both test software and Kiwire are using the same type of hardware and mikrotik switches .



Logical diagram of test lab setup.

4. Server Details

Server Model	Dell R210
CPU	1 socket dual core
CPU model	
Ram	16 GB
Harddisk	Raid 1 500 GB
Operating System	Redhat Enterprise 6.5 Standard edition
Database	MySQL 5

5. Performance objective

The objective of this benchmark performance test in our lab is to perform and evaluate kiwire handle to stress load of 10,000 user as benchmark.

Kiwire consist of few component mainly

- Apache web server
- Radius services
- MySQL database engine

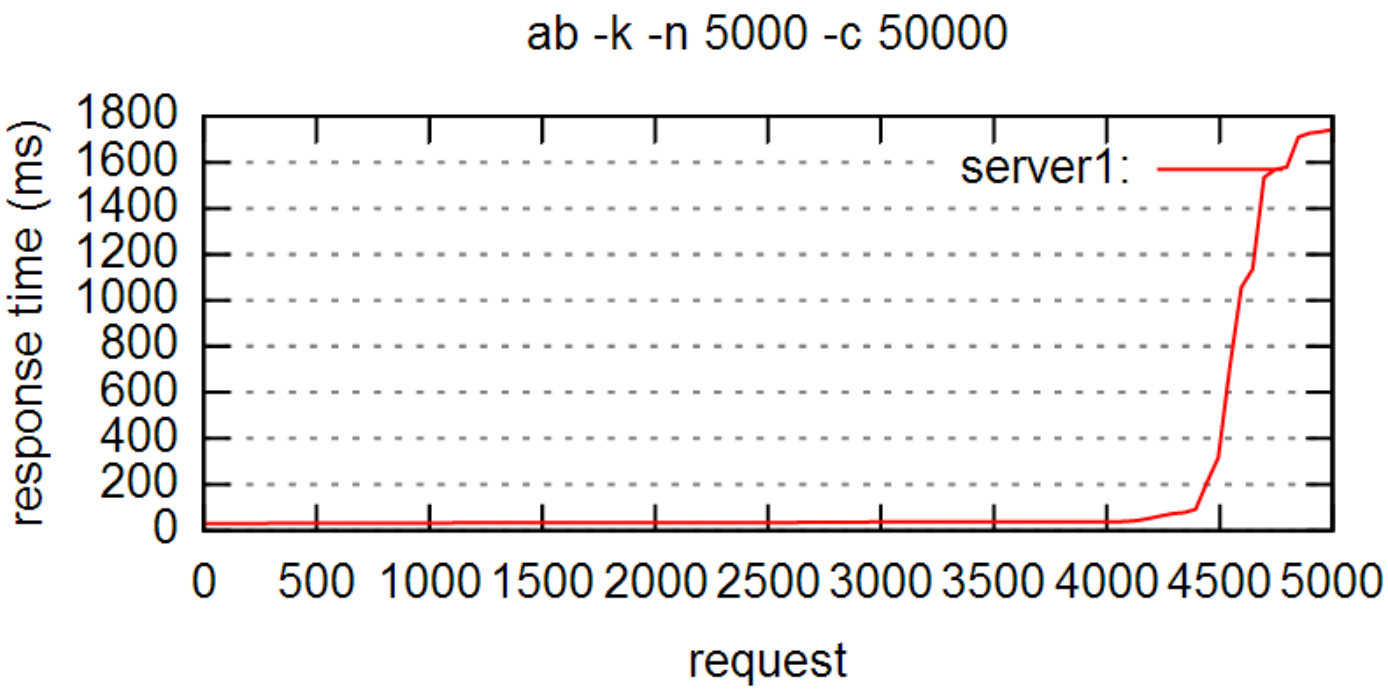
As the apache and radius are loading data from mysql every benchmark will directly load test the MySQL database this will ensure the mysql is being benchmark as well.

5.1. Apache Benchmark

The tools we use will be apache “ab” .ab is a tool for benchmarking Apache Hypertext Transfer Protocol (HTTP) server. It is designed to give an impression of how current Apache installation performs. This especially shows how many requests per second Apache installation is capable of serving.

5.1.1. Benchmark 1

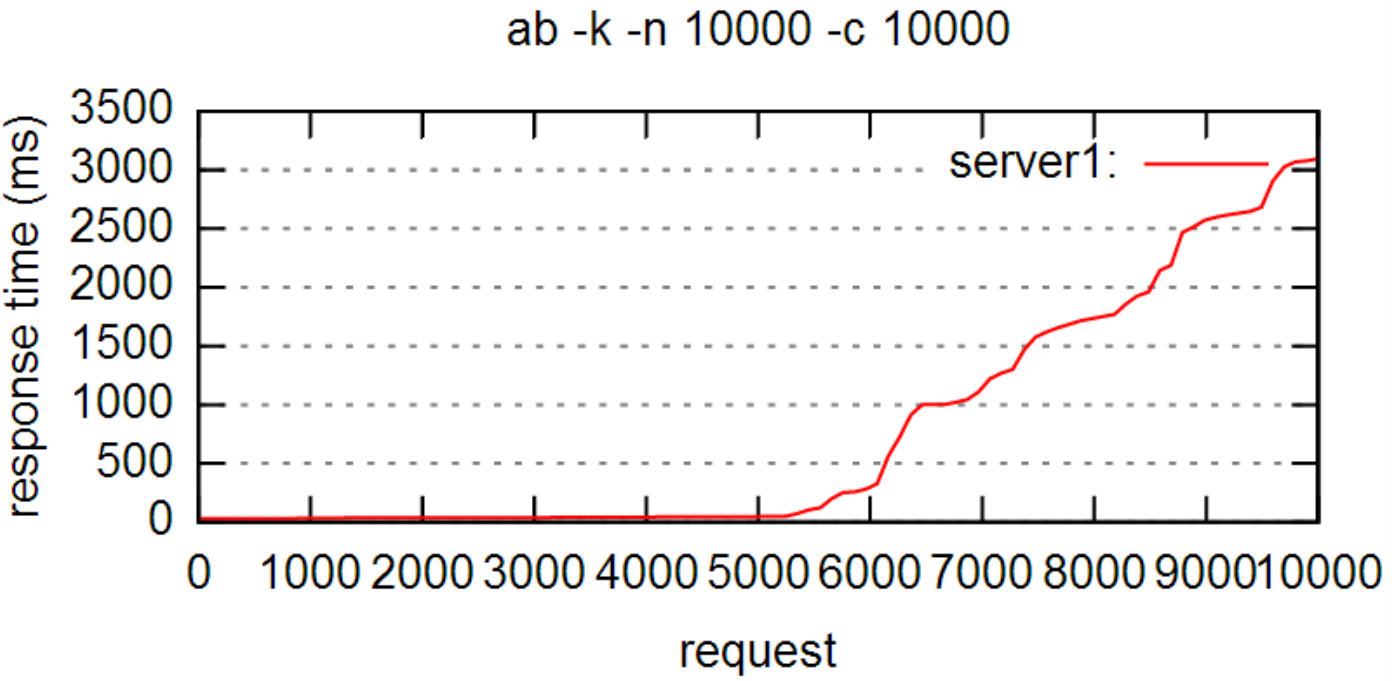
On benchmark 1 , we be doing a test of 5000 request with 5000 concurrency at 1 burst.



```
This is ApacheBench, Version 2.3 <$Revision: 655654 $>
Copyright 1996 Adam Twiss, Zeus Technology Ltd, http://www.zeustech.net/
Licensed to The Apache Software Foundation, http://www.apache.org/

Benchmarking 192.168.2.9 (be patient)

Server Software:        Apache/2.2.15
Server Hostname:        192.168.2.9
Server Port:            80

Document Path:          /
Document Length:        2017 bytes

Concurrency Level:      5000
Time taken for tests:   1.761 seconds
Complete requests:      5000
Failed requests:        0
Write errors:           0
Non-2xx responses:      5003
Total transferred:      11314344 bytes
HTML transferred:       10088609 bytes
Requests per second:    2838.68 [#/sec] (mean)
Time per request:       1761.380 [ms] (mean)
Time per request:       0.352 [ms] (mean, across all concurrent requests)
Transfer rate:          6273.02 [Kbytes/sec] received

Connection Times (ms)
              min  mean[+/-sd] median   max
Connect:        2   78 256.1      6    1041
Processing:    10   98 243.2     30    1552
Waiting:       10   96 243.0     28    1552
Total:         31  176 425.8     36    1742

Percentage of the requests served within a certain time (ms)
  50%     36
  66%     38
  75%     38
  80%     39
  90%    285
  95%   1569
  98%   1727
  99%   1732
 100%   1742 (longest request)
```

5.1.2. Benchmark 2

On bechmark 2 we perform a test with 10,000 request with 10,000 concurrency at 1 go this simulate a flood user of user.




```
Concurrency Level:      10000
Time taken for tests:   3.135 seconds
Complete requests:      10000
Failed requests:        0
Write errors:           0
Non-2xx responses:      10014
Total transferred:      22640272 bytes
HTML transferred:       20186842 bytes
Requests per second:    3190.03 [#/sec] (mean)
Time per request:       3134.764 [ms] (mean)
Time per request:       0.313 [ms] (mean, across all concurrent requests)
Transfer rate:          7053.05 [Kbytes/sec] received

Connection Times (ms)
              min  mean[+/-sd] median   max
Connect:        1  445 716.5      7    3047
Processing:     1  314 514.1     37    2771
Waiting:        1  310 511.9     33    2771
Total:         29  760 988.3     46    3097

Percentage of the requests served within a certain time (ms)
  50%     46
  66%   1002
  75%   1597
  80%   1736
  90%   2575
  95%   2690
  98%   3068
  99%   3078
 100%   3097 (longest request)
```

5.1.3. Summary

The load test shown that the system is able to witstand 10,000 connection incoming without any error and time taken to furfill the load of web portal page is within 3.4 second. The time to load will increase as the load increases which is acceptable between normal parameter.

5.2. Radius Server Load test

The Radius engine is an RFC compliant enterprise radius engine. For the test we be using linux radclient which can send no of requests in parallel to the radius server.
As radius is already an robust services we will perform multi-thread load test which will will initiate the program 10 instance with 1000 connection for start to simulate 10,000 connection to the system. Command used is "*time echo "User-Name=test,Password=test" | radclient -c 1000 -p 1000 -q -s 192.168.2.9:1812 auth testing123*"

5.2.1. Benchmark 1

The first benchmark is 10,000 connection request via 10 instance with 1000 request each. Which checking if there is any lost-auth which stand for server unable to response or timeout .

Instance with 1000 request in parallel	Time to complete (in second)
1	117.7
2	118.3
3	118.6
4	117.1
5	117.9
6	119.1
7	118.4
8	117.7
9	118.4
10	118.1
Average	**118.13 / 1 min 58 second**

Lost Auth : 0

5.2.2. Benchmark 2

20,000 connection via 20 instances.

Instance with 1000 request in parrallel	Time to complete in sec
1	145.4
2	146.9
3	148.8
4	145.9
5	146.9
6	147.9
7	149.2
8	150.2
9	151.4
10	149.4
11	148.0
12	149.6
13	148.8
14	149.5
15	148.6
16	148.8
17	148.0
18	148.1
19	148.4
20	148.7
21	150.7
22	149.9
23	149.6
24	149.8
25	147.8
26	147.5
27	148.4
28	147.5
29	148.3
30	148.2
Average	**148.54 / 2 min 28.54 second**

Lost Auths : 0

6. Stress test , benchmark summary.

From the load test performed its safe to conclude the kiwire can safely handle 10,000 request flood and able to compute it within 2.5 minutes with zero fault and error .

This is a result of our development culture of utilize coding techniques and good programming practices to create high quality code plays an important role in software quality and performance. In addition, by consistently applying a well-defined coding standard and proper coding techniques, and holding routine code reviews.

Note:
This test is performed in an full kiwire enviroment without any 3rd party api or interface involved , as 3rd party api are not part of kiwire standard application services offering.

The test is perform on a local area network which network lattency will not affect our performance. In real world the performance result may defer due to end-user network enviroment.