**CSC326**

**Lab 4 Report**

**Development Phase 4**

Group #: 4

Names: Anny Kim Ly (997581718)

Kevin Gumba (997585117)

**Site:**

with HAproxy

ec2-54-204-43-178.compute-1.amazonaws.com

without HAproxy

<http://ec2-54-204-43-174.compute-1.amazonaws.com/>

[http://ec2-54-225-76-58.compute-1.amazonaws.com](http://ec2-54-225-76-58.compute-1.amazonaws.com/)

**1.0 Metrics**

In order to test the front and backend of the server, Apache benchmarking is used. Apache benchmarking allows to monitor the network, CPU, and memory usage of the web instance.

**2.0 Bottlenecks and Improvements**

***2.0.1 HAproxy***

The server is set up with HAproxy as the load balancer to 2 frontend instances. Using a load balancer allows for many clients to access the front end server without it crashing. In addition, 2 frontend instances are used in order to allow more client requests.

Through the Apache benchmarking, it is evident that the server can only hold up to approximately 40 clients at a time before crashing. A more efficient server should be able to hold more clients, of up to 1000 at a time. In order to handle this bottleneck, HAproxy is used. HAproxy acts as a load balancer to transfer user requests to different instances of the web page. This allows for more users to access the server at the same time.

Refer to Data 1 to 4 to see improvements of the server from the old CPU and memory usage to the new results.

**2.0.2 Reduce update repetition**

Before lab 3, the document and URL ID is continually updated everytime the same web page is traversed. Repeatedly updating variables when not needed is time consuming, especially when it is much desirable for crawler.py to finish quickly. In order to delete repetition of code, a simple if statement is added so that the ID is only updated once.

**2.0.3 Reduce Disk I/O Access**

Disk I/O Access contributes greatly to the bottlenecks of the search engine. For each keywords, the Front End must retrieve the results from the database stored in memory. To reduce this, caching is implemented in multiple areas.

**2.0.3a Top Keywords - Preload and Caching**

On initial loading of the results page, the results of the top 20 keywords are retrieved and preloaded into the cache. If the user searches for a keyword that is one of the top 20 keywords, then the results can be retrieved from the cache instead of from disk.

**2.0.3b Browser Caching**

During each session, the search results are temporarily stored in the cache. At the end of the session, the cache is cleared. If the user searches for a keyword that was previously searched in that session, then the results can be retrieved from the cache instead of from disk.

**2.0.3c Template Caching**

Making use of Bottle’s template engine, the web pages are stored as templates in separate files having a .tpl extension. Templates are usually compile only once and cached internally, but can be rendered many times with different keywords arguments.

**3.0 Performance Results**

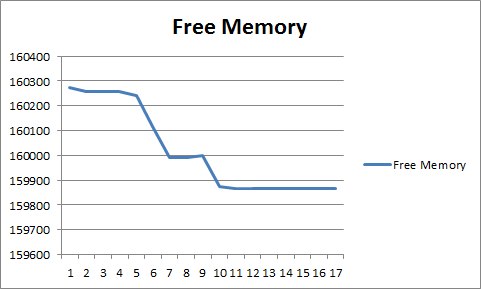
By running 1010 connections using apache, the server is able to handle all requests. CPU only drops to approximately 80% while in our previous version, drops to almost 70% (Refer to Data 1 and 2 graphs related to CPU - % idle). Running with 1010 requests, less memory is used over time as a result of caching (Refer to Data 3 and 4).

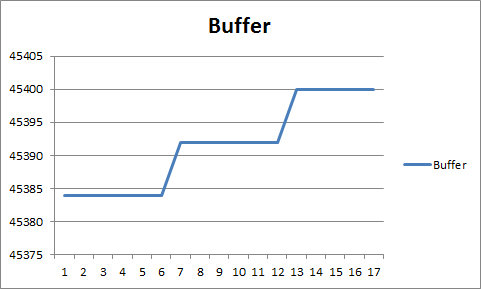
**Data 1:** Previous version (lab 3)

Running -n 2000 -c 10

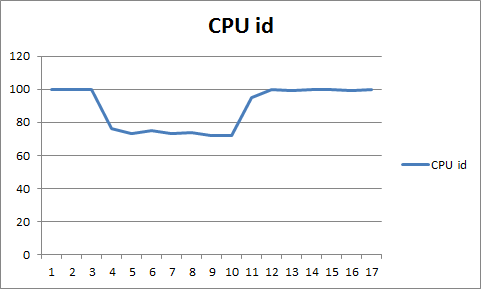
Note: X axis is in seconds.

**Memory Usage**

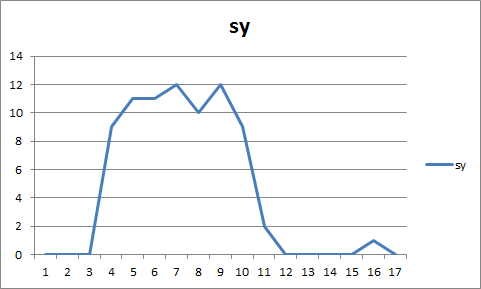




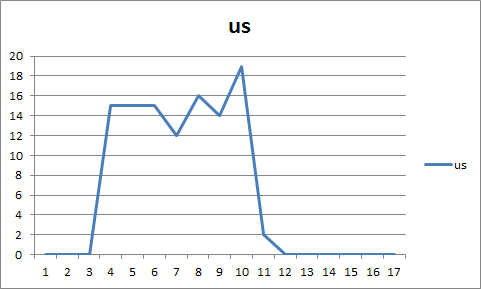
**CPU Usage**



% of time CPU is idle



% of time CPU spent in kernel mode

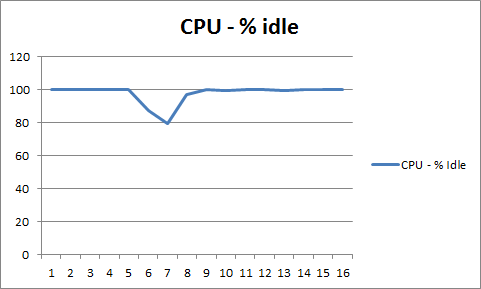


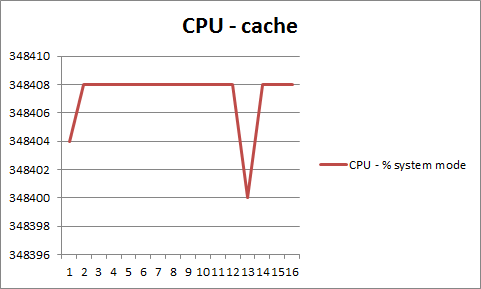
% of time CPU spent in user mode

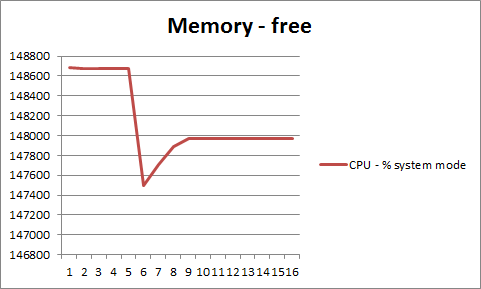
**Data 2:** Lab 4

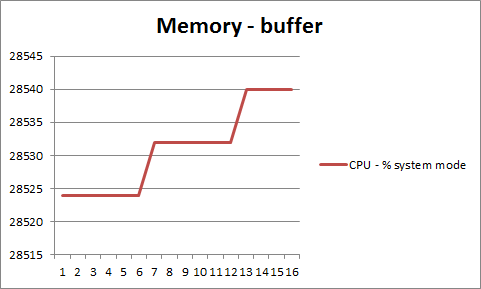
ab -n 1010 -c 1010 http://ec2-54-204-43-178.compute-1.amazonaws.com/

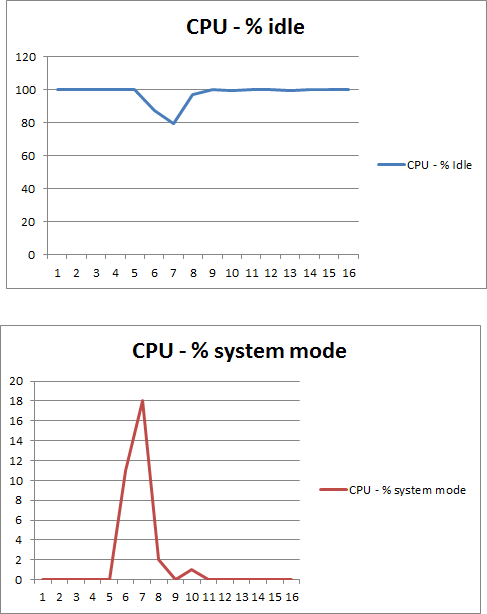
Note: x-axis in seconds











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 ec2-54-204-43-178.compute-1.amazonaws.com (be patient)

Completed 101 requests

Completed 202 requests

Completed 303 requests

Completed 404 requests

Completed 505 requests

Completed 606 requests

Completed 707 requests

Completed 808 requests

Completed 909 requests

Completed 1010 requests

Finished 1010 requests

Server Software: WSGIServer/0.1

Server Hostname: ec2-54-204-43-178.compute-1.amazonaws.com

Server Port: 80

Document Path: /

Document Length: 4898 bytes

Concurrency Level: 1010

Time taken for tests: 2.786 seconds

Complete requests: 1010

Failed requests: 0

Write errors: 0

Total transferred: 5103530 bytes

HTML transferred: 4946980 bytes

Requests per second: 362.57 [#/sec] (mean)

Time per request: 2785.655 [ms] (mean)

Time per request: 2.758 [ms] (mean, across all concurrent requests)

Transfer rate: 1789.14 [Kbytes/sec] received

Connection Times (ms)

min mean[+/-sd] median max

Connect: 17 30 7.5 30 43

Processing: 46 680 438.7 554 2729

Waiting: 45 679 438.7 553 2728

Total: 63 710 444.4 591 2772

Percentage of the requests served within a certain time (ms)

50% 591

66% 881

75% 994

80% 1043

90% 1135

95% 1751

98% 1777

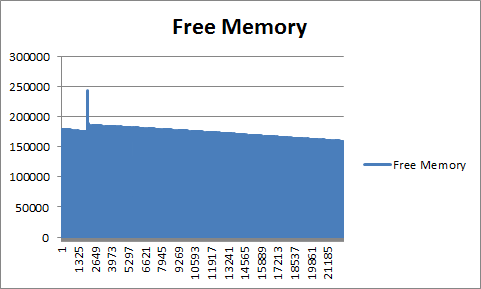
99% 1987

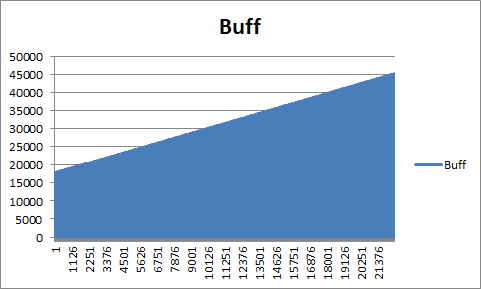
100% 2772 (longest request)

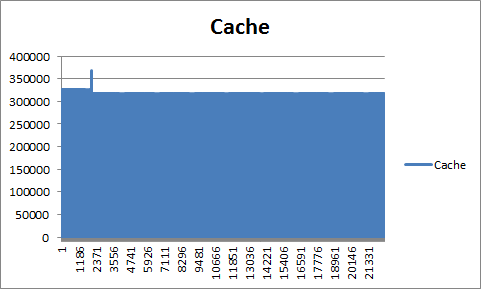
**Data 3:** Running with 100000 requests

Note: X axis is in seconds.

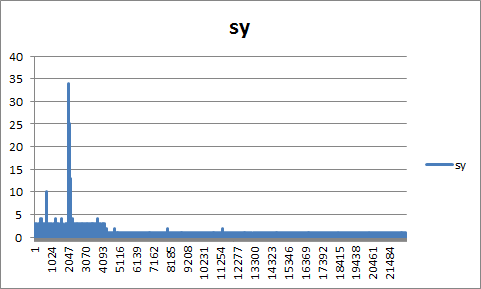
**Memory Usage**



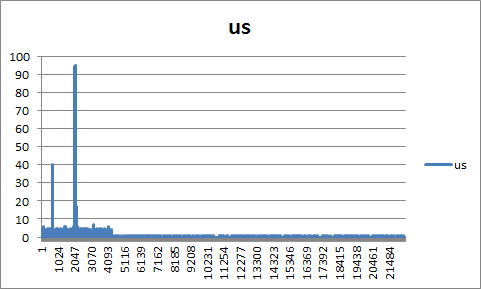




**CPU Usage**



% of time CPU spent in kernel mode

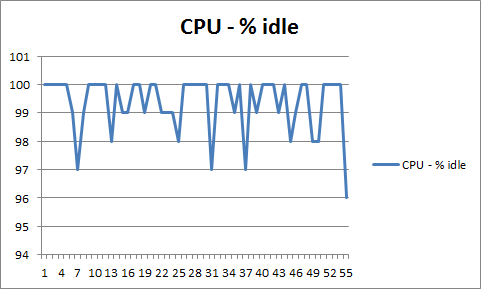


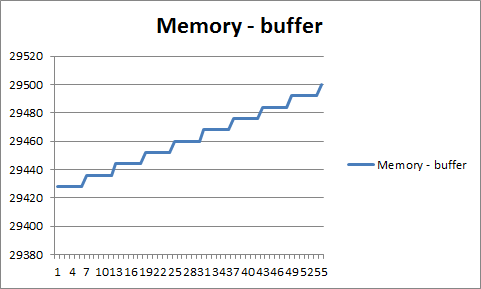
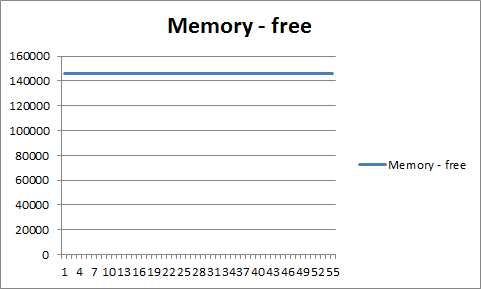
% of time CPU spent in user mode

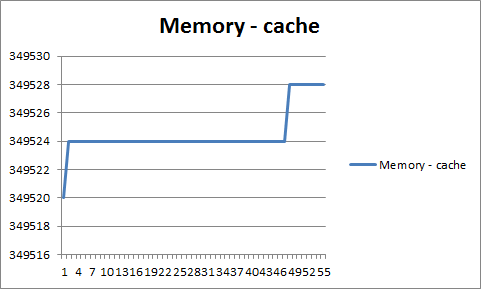
**Data 4:**

ab -n 1010 <http://ec2-54-204-43-178.compute-1.amazonaws.com/>

Note: x-axis in seconds







ab -n 1010 http://ec2-54-204-43-178.compute-1.amazonaws.com/

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 ec2-54-204-43-178.compute-1.amazonaws.com (be patient)

Completed 101 requests

Completed 202 requests

Completed 303 requests

Completed 404 requests

Completed 505 requests

Completed 606 requests

Completed 707 requests

Completed 808 requests

Completed 909 requests

Completed 1010 requests

Finished 1010 requests

Server Software: WSGIServer/0.1

Server Hostname: ec2-54-204-43-178.compute-1.amazonaws.com

Server Port: 80

Document Path: /

Document Length: 4898 bytes

Concurrency Level: 1

Time taken for tests: 37.666 seconds

Complete requests: 1010

Failed requests: 0

Write errors: 0

Total transferred: 5103530 bytes

HTML transferred: 4946980 bytes

Requests per second: 26.81 [#/sec] (mean)

Time per request: 37.293 [ms] (mean)

Time per request: 37.293 [ms] (mean, across all concurrent requests)

Transfer rate: 132.32 [Kbytes/sec] received

Connection Times (ms)

min mean[+/-sd] median max

Connect: 17 17 0.2 17 19

Processing: 19 20 3.8 20 52

Waiting: 19 20 3.7 19 51

Total: 36 37 3.8 37 69

Percentage of the requests served within a certain time (ms)

50% 37

66% 37

75% 37

80% 37

90% 37

95% 39

98% 49

99% 63

100% 69 (longest request)

Problem Encountered

The front-end relies on passing variables between URL pages. After combining ideas on improving server performance through load balancing and caching, it is later found that HAproxy does not allow passing variables between pages efficiently: After combining the HAproxy and caching together, an error occurs on the results page. At random times, when a user searches up a new keyword, the results from the previous search is kept for the next few minutes. The search engine then starts working properly again, however the problem can be re-encountered.

“The Domain Name System (DNS) resolves the URI to one of the static IP addresses associated with that fully qualified domain name. (All load-balancer servers share the same fully qualified domain name but have unique static IP addresses.)[”](http://www.google.com/url?q=http%3A%2F%2Fsupport.rightscale.com%2F06-FAQs%2FFAQ_0101_-_What_is_HAProxy_and_how_does_it_work%253F&sa=D&sntz=1&usg=AFQjCNFAFBgDCA4eTwNKn_bz0KxWQ3FI5g) (<http://support.rightscale.com/06-FAQs/FAQ_0101_-_What_is_HAProxy_and_how_does_it_work%3F>)

In order to bypass this, when the user searches a keyword and is on the results page, in order to query another search, he/she must press the back button and return to the previous page and do the search from there.

The search engine without the HAproxy (with caching implemented) can be accessed through these URLs:

<http://ec2-54-204-43-174.compute-1.amazonaws.com/>

[http://ec2-54-225-76-58.compute-1.amazonaws.com](http://ec2-54-225-76-58.compute-1.amazonaws.com/)