S15 15619 Project Phase 2 Report

Performance Data and Configurations

<table>
<tr><th>Best Configuration and</th><th>Results from the Live Test</th></tr>
<tr><td>Number and type of instances</td><td>HBase Live Test: 9 m1.large
MySQL Live Test:8 m1.large</td></tr>
<tr><td>Cost per hour
(assume on-demand prices)</td><td>HBase Live Test: 1.75
MySQL Live Test:1.56</td></tr>
<tr><td>Queries Per Second (QPS)</td><td>INSERT HERE: (Q1,Q2H,Q2M,Q3H,Q3M,Q4H,Q4M)
<table>
<tr><td></td><td>Q1</td><td>Q2H</td><td>Q2M</td><td>Q3H</td><td>Q3M</td><td>Q4H</td><td>Q4M</td></tr>
<tr><td>score</td><td>120.39</td><td>120.39</td><td>131.6</td><td>84.77</td><td>109.42</td><td>8.78</td><td>175.24</td></tr>
<tr><td>tput</td><td>18058.0</td><td>18058.0</td><td>7896.0</td><td>8650.4</td><td>11164.8</td><td>543.6</td><td>10514.2</td></tr>
<tr><td>ltcy</td><td>5</td><td>5</td><td>6</td><td>5</td><td>4</td><td>91</td><td>4</td></tr>
<tr><td>corr</td><td>100.00</td><td>100.00</td><td>100.00</td><td>100.00</td><td>98.00</td><td>100.00</td><td>100.00</td></tr>
<tr><td>err</td><td>0.00</td><td>0.00</td><td>0.00</td><td>0.00</td><td>0.00</td><td>0.00</td><td>0.00</td></tr>
</table>
</td></tr>
<tr><td>Rank on the scoreboard:</td><td>Q1: 13
Q2H: 9
Q2M: 5
Q3H: 3
Q3M: 10
Q4H: 17
Q4M: 6
HBase Live Test: 8
MySQL Live Test: 8</td></tr>
</table>

Team : Oak
Members : Ziyuan Song, Aaron Hsu, Jiali Chen

Rubric:

Each unanswered question = -7%
Each unsatisfactory answer = -3%

Task 1: Front end (you may/should copy answers from your earlier report-- each report should form a comprehensive account of your experiences building a cloud system. Please try to add more depth and cite references for your answers from the P1Report)

Questions

1. **Which front end framework did you use? Explain why you used this solution.** [Provide a small table of special properties that this framework/platform provides].

We used Undertow after investigation to see which front end frameworks has the most efficiency in terms of RPS. This is because on EC2 instances, it has one of the highest performance with both plain text and query throughputs. These can be found on https://www.techempower.com
Undertow is lightweight, embedded, flexible, as argued on numerous sources on the web. It has both synchronous and asynchronous (based on NIO) APIs. It allows multiple combinations of various handlers which can be easily added when needed, and removed when trying to get rid of extra overheads. (EDIT) In addition, it allowed pipelining, using the dispatch function from undertow, we were able to thread multiple connections to both mysql and hbase, thus allowing us to use a single front end for a backend server.

2. **Did you change your framework after Phase 1? Why or why not?**

No we did not. We felt that the undertow framework is fast and reliable enough. Since we were able to reach 17000 RPS with only one front end for Q1, our major concerns were not about the front end framework. Our primary bottleneck resides at the backend, which we spent most of our time tuning.

3. **Explain your choice of instance type and numbers for your front end system.**

For Mysql, we used 4 instances of M1.Large instance type. We chose m1 instances over m3 because m1 instances were rumored to be more effective at PV. In addition, we chose large instances over medium and small instances to avoid overheads caused by ELB round robin switching. Also, we chose the minimum number of m1.large instances required to pass

15000 RPS because we wanted to save money FOR YOU GUYS.We were under

the limit of 1.25 dollars for Q1, using almost only 8% of the limit.
(EDIT) However after further investigation in Q2, we found out that m3.large is more efficient than m1.large because it has faster processing power, and is cheaper in terms of on-demand price. In Q2, for mysql, we had 3 m3.large instances, and 1 m3.large instance for hbase.
(EDIT) During phase 2, we decided to go with 4 m1.large instances for mysql and 1 m1.large instance for hbase. This was due to our infrastructure of 4 backend servers for mysql and a

master node for hbase. However, we realized that ELB does not perform well for mysql Q1. So with hindsight we, should have just provisioned a backend with 20k IOPS and a single frontend without an ELB. Then we would be able to reach the desired score for Q1 and at the same time reaching the scores we got for other Q's.

4. **Explain any special configurations of your front end system.**

We used an Amazon paravirtualized machine (since PV should be better than HVM in terms of efficiency),configured the instance to support java, undertow and produced an AMI based on that configuration. This allows us to easily create instances running the front end web-servers when horizontal autoscaling is needed.

We configured our front end to support ordered pipelining request to our backend. (This was done by undertow dispatch method) This allows our one single front end to have multiple connections to a back end, so instead of having one backend and multiple front ends (with back end being the bottleneck), we can have n number of replicated backends paired with n number of front ends (In mysql).

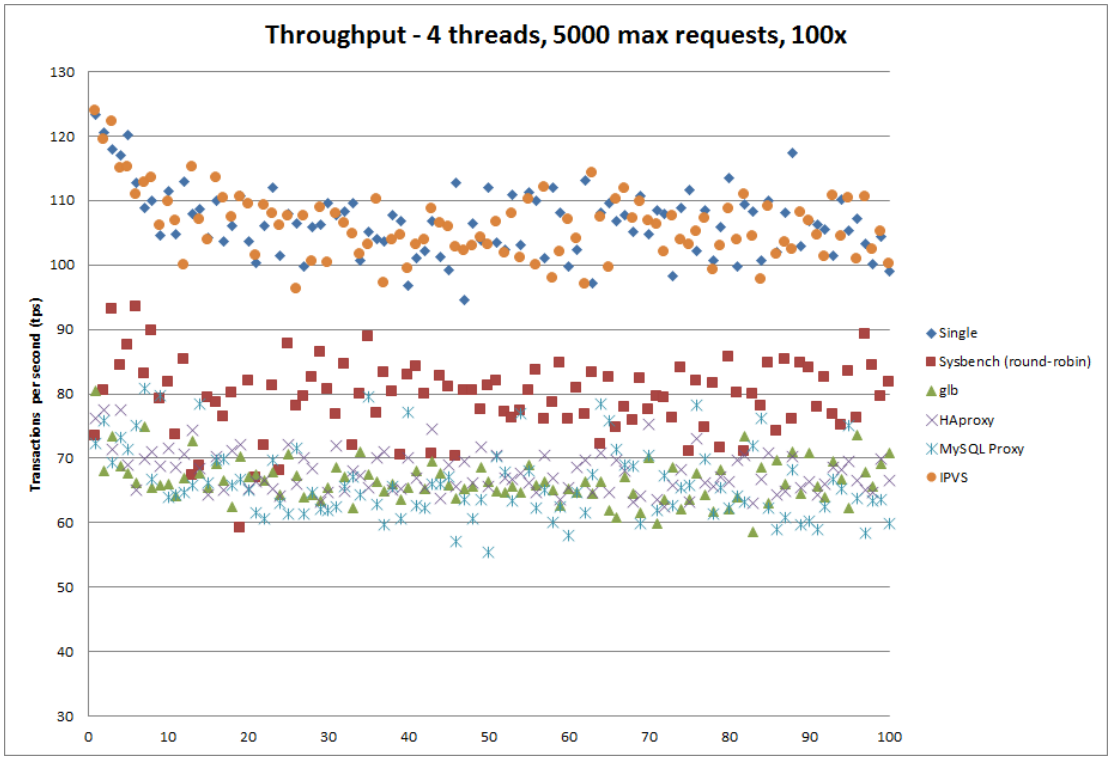
5. **Did you use an ELB for the front-end? Why, or why not? Condense your experience with ELB in the next few sentences. Talk about load-balancing in general and why it matters in the cloud.**

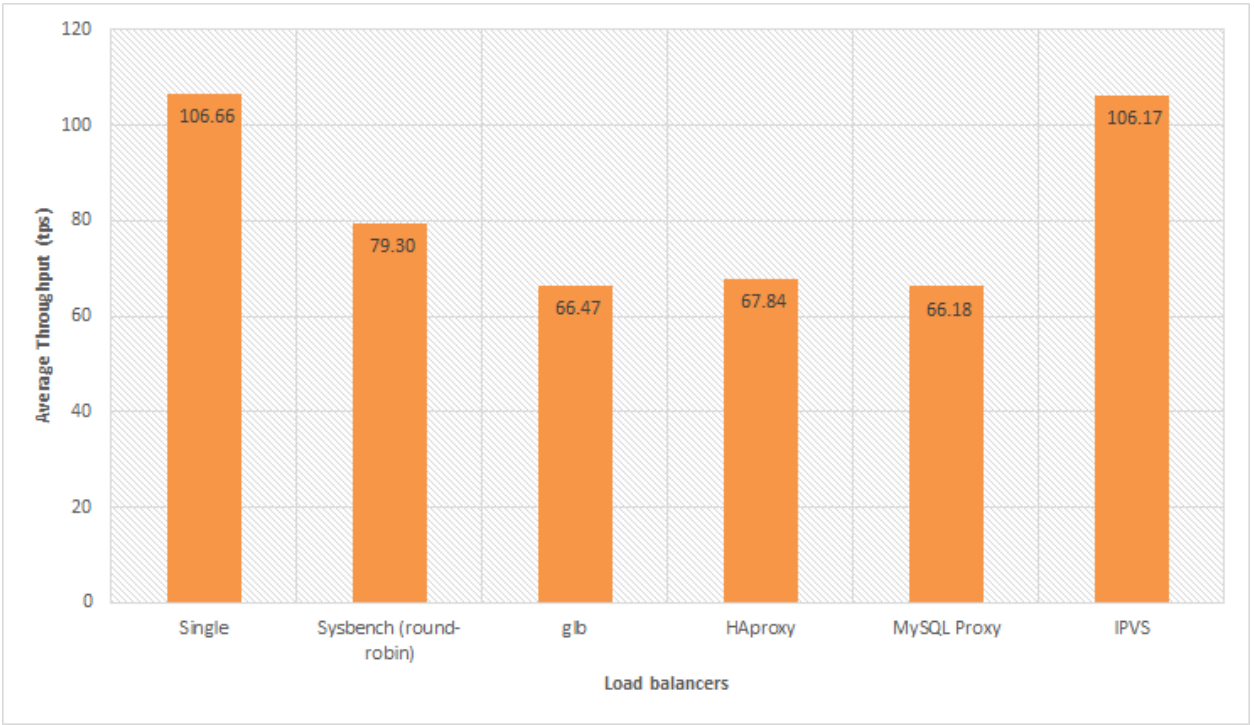
Yes, we used an ELB for the front-end. Because ELB can automatically route traffic across multiple instances to achieve higher levels of fault tolerance, it also automatically scales its request handling capacity to meet the demands of application traffic, since we had multiple front ends for achieving high RPS. It reduces the bottleneck by spreading the traffic to multiple front ends, routing to multiple backends. However, after phase 2, we found out that ELB does not perform well with multiple front end queries for Q1. This was because each frontend does its own threading, but ELB distributes the queries to each threading frontend, so the Q1 score was lower than expected.

6. **Did you explore any alternatives to ELB? List a few of these alternatives. What did you finally decide to use? (if possible) Provide some graphs comparing performance between different types of systems.**

In phase 1, we tried using Haproxy instead of an aws ELB. Unlike the ELB, Haproxy is not a black box, since we were able to see the exact traffic and ports opened to each front end. In addition, it does not require warming up. Prior to warming up, Haproxy performs much better than ELB, however after warming up the ELB, it allows a much higher throughput than Haproxy. (For mysql, Haproxy achieves around 6000 with 2 backend, 2front end, while ELB achieves around 6500 after warming up.) So in phase2, we only use ELB which provides more stable performance.

The two graphs below show two normals(single and sysbench), with three layer 7 load balancers and one layer 4 switcher. Since we wanted to finish the project before investing more time for the complicated IPVS switcher, we decided to use ELB first before the deadline for phase 2. We will most likely test out IPVS in phase 3.



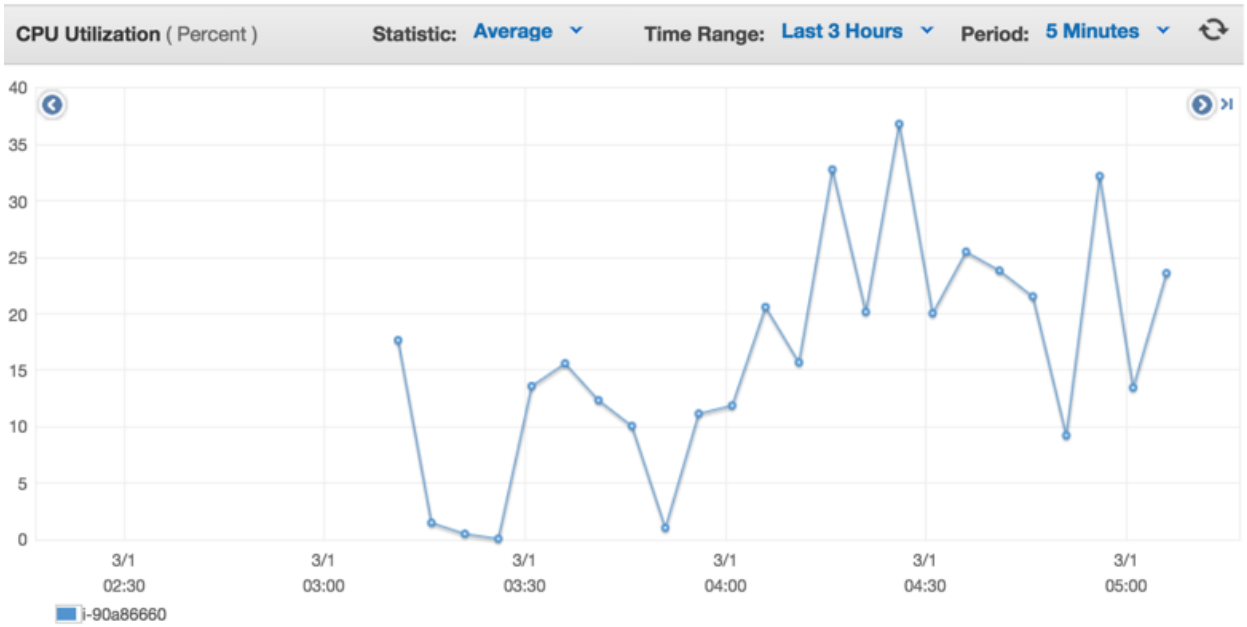


7. **Did you automate your front-end instance? If yes, how? If no, why not?**

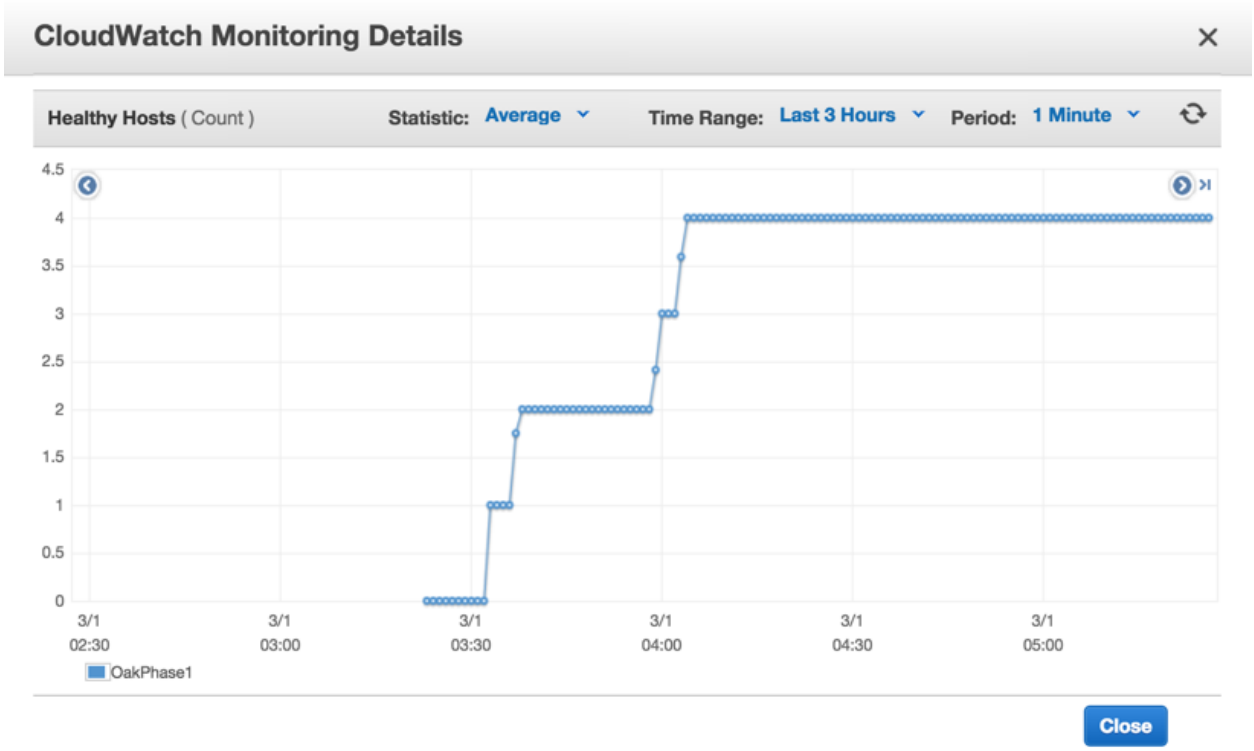
Yes, we wrote a shell script that updates, installs all required packages and added the script that initiates the front end server. Then we added the command that initiates the front end server to crontab, specifying that the server should be initiated when system startup. Then, we created an amazon machine image with it, so then we'll be able to start multiple instances with the server running automatically. (For Autoscaling purposes that may be required in the future?).

8. **Did you use any form of monitoring on your front-end? Why or why not? If you did, show us a capture of your monitoring during the Live Test.** Else, try to provide CloudWatch logs of your Live Test in terms of memory, CPU, disk and network utilization. Demarcate each query clearly in the submitted image capture.

Yes, we used CloudWatch metrics: Basic monitoring. Because we want to monitor the cpu utilization of front-end instance to check whether we need more instances to meet the demands of traffic. In addition, we also used 'top' to monitor %wa (io wait time), and %id (remaining CPU), and 'free' for checking available RAM so we can decide how much for and whether if swapping is occurring.



We also check the health status of instances to guarantee that every instance is running and receiving url requests.



9. **What was the cost to develop the front end system?**

In phase 1 it was 17.54+2.85+0.748 = 21.138
In phase 2, we barely allocated any time for tuning our front end system.

10. **What are the best reference URLs (or books) that you found for your front-end? Provide at least 3.**

http://javaarm.com/faces/display.xhtml;jsessionid=j8CbEIX-4bB-B+6C0VcaxRK4?tid=3314&page=1&print=true

http://undertow.io/documentation/core/undertow-request-lifecycle.html

http://repository.jboss.org/nexus/content/unzip/unzip/io/undertow/undertow-core/1.0.14.Final/undertow-core-1.0.14.Final-javadoc.jar-unzip/io/undertow/server/HttpServerExchange.html?nsukey=4rb5LcQ%2Fm%2BVQcSWTkLfVV1wpxMYKHAyXN%2BxmK%2BHWoI066%2BPFeB2jYsoFpJh0IbWimFSSRyEVyLqGZNrYX7mysA%3D%3D

https://www.youtube.com/watch?v=ZZ5LpwO-An4

[Please submit the code for the frontend in your ZIP file]

Task 2: Back end (database)
Questions

1. **Describe your schema for each DB. Explain your schema design decisions. Would your design be different if you were not using this database? How many iterations did your schema design require? Also mention any other design ideas you had, and why you chose this one? Answers backed by evidence (actual test results and bar charts) will be valued highly.**

Since our task was to return the censored text and a calculated score from user id and time. We transformed the data with mapreduce such that the row key for our databases was userid+time, and the value would be the tweetid+calculated score+censored text. This allows the front end to have almost no extra calculation. We only needed one mapreduce to extract transform the data, and one load each for mysql and hbase.

For **mysql**, our first column (userid+time) was not unique. At first, we were hesitant about which engine to use, innoDB or Myisam. However, after closer evaluation, we chose to use Myisam because Myisam has several optimization options that specializes for reading operations. The first option was a fixed row optimization, this allows reading simply by pointing at the correct block, thus allowing really fast reads. The benchmark could be found here
http://www.soliantconsulting.com/blog/2012/09/mysql-optimization-faster-selects-myisam-fixed-row-format
However, this requires that we change the text(tweetid+calculated score+censored text) to be a fixed length (max of text length). This requires too much disk space, so we disregarded it. The option we chose was myisam compression + indexing. After we loaded the data into the MYD file, we indexed them with our row key. Then we myisampack-ed them compressing the MYD file and resorted the MYI index file. This enables the database to perform less IO disc reads to find the query data.
http://www.techrepublic.com/article/save-disk-space-by-compressing-mysql-tables/

For **Hbase,** our use importTsv+bulk load to load data to hadoop HBase. This will be more efficient to load large data to HBase database, which naturally support "\t" separator and mapreduce.
http://hbase.apache.org/book.html#importtsv
Then we use "family" as family name, which we know is better to be short like 'f' and the same with cells, like 'c1', 'c2', and so forth. We alter our HBase big table with compression via "SNAPPY", "Bloomfilter" set by ture and IN_MEMORY set by false.
http://www-01.ibm.com/support/knowledgecenter/SSPT3X_3.0.0/com.ibm.swg.im.infosphere.biginsights.analyze.doc/doc/bigsql_TuneHbase.html

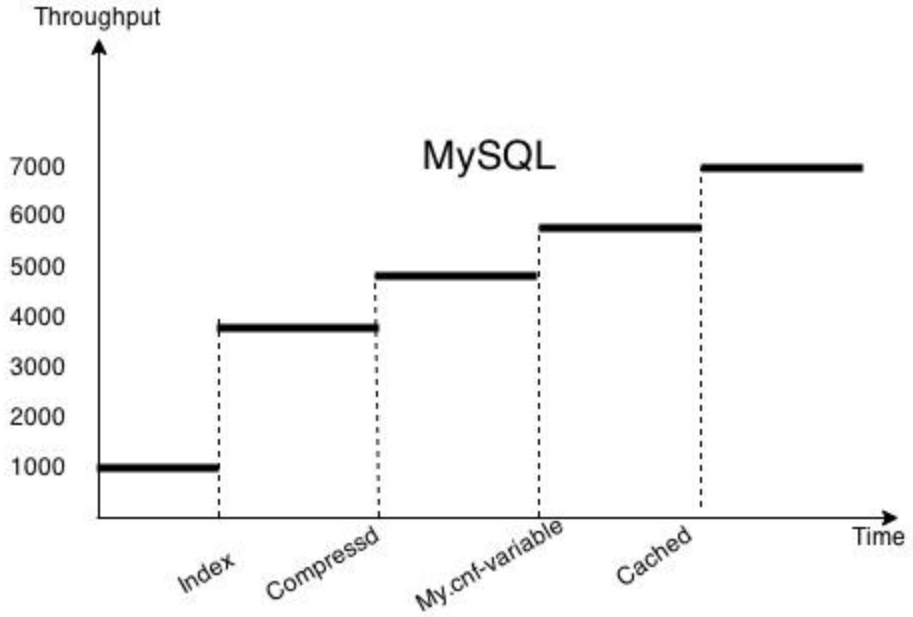
(EDIT)In phase 2, for mysql, we used innodb instead of myisam. This was because that we found out that innodb supports threading and multiple connections better. It was because innodb uses threading more efficiently than myisam. One of the biggest thing we didn't

further investigate in was innodb compression. This was due to time restraints and our decision was to tune hbase more.
http://web.performancerasta.com/data-analytics-myisam-vs-archive-vs-infobright-ice-vs-infobright-iee-benchmark/
Most importantly
http://www.lemug.fr/wp-content/uploads/2011/01/Wp_MySQL-5.5_InnoDB-MyISAM.en_.pdf

In phase 2, for mysql, both Q2 and Q3 had the same table design, it was a simple row key that points to a value. However, for Q4, we had to design the table so that we were able to perform one single row get to gain access to all dates and values. Thus for each column family, we had all dates as each column (qualifier). This enabled us to access all dates for a hashtag really quickly.

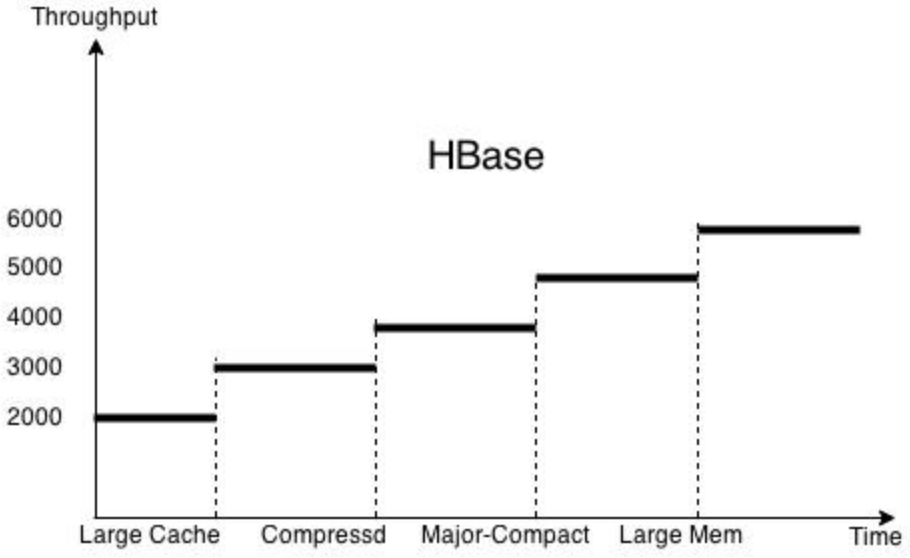
2. **What was the most expensive operation / biggest problem with each DB that you had to resolve for each query? Why does this problem exist in this DB? How did you resolve it? Plot a chart showing the improvements with time.**

For MySQL, the most expensive operation was definitely disc IO. This could be seen from the 'top' command and looking at %wa when the database is under load from the load generator. This problem exists in this DB because of the intensive read operations. Thus we chose to compress the data with myisampack, and resorted the index after compression.



For Hbase, the most expensive operation was definitely "hot region". We do major-compact to make data to be locality. Also we change to compress data by "SNAPPY" and enhance the

block cache. All of these is to decrease IO disc reading and improve memory efficiency. When we use HBase, it generated sharding regions by its own(we do not use split when we create table. We guess it may work if we create a table with split by split points, which can help to resolve hot region problem). So if the querying is within a specific range, which is stored only in one region, it will result in a “hot region” troubleshooting. There are ways to resolve it. One is to split a hot region into parts and move to different servers. The other is to improve IO speed. We try these two ways, and the latter one has a better improvement on performance.



In phase 2, for mysql, the biggest problem was IOPS quota. Since we couldn’t cache anything due to the nature of random queries, we had to use provisioned IOPS SSD.
As for hbase, our biggest problem was fine tuning each query to reach the desired RPS. There are a lot of areas we still need to investigate in.

3. **Explain (briefly) the theory behind (at least) 7 performance optimization techniques for databases. How are each of these implemented in MySQL? How are each of these implemented in HBase? Which optimizations only exist in one type of DB? How can you simulate that optimization in the other (or if you cannot, why not)? Use your own words (paraphrase, this document goes through plagiarism detection software).**

    a. Replication - This allows different front ends to be able to connect to their own database, reducing the bottleneck of the backend if it is one. So multiple read queries can be done simultaneously without accumulating CPU and IO-wait time.
    b. Sharding - Splits the one database into several smaller databases. This allows read and write operations to be performed parallely at the smaller databases, reducing time spent waiting on read/write locks. This also reduces the

number of look ups needed to find the data for a query.

c. Indexing - Stores pointers to data that can be found with faster look up speed. For instance the indexes can be store via hash table or Btree.

For MySQL, replication and sharding is done via a mysql cluster. An API node would be the communication node between front ends and all its data nodes, while a master node allows configuration of the mysql cluster. A user could specify the number of sharding or replication nodes through the master node. As for indexing, the engine we used (myisam) allows only Btree indexing.

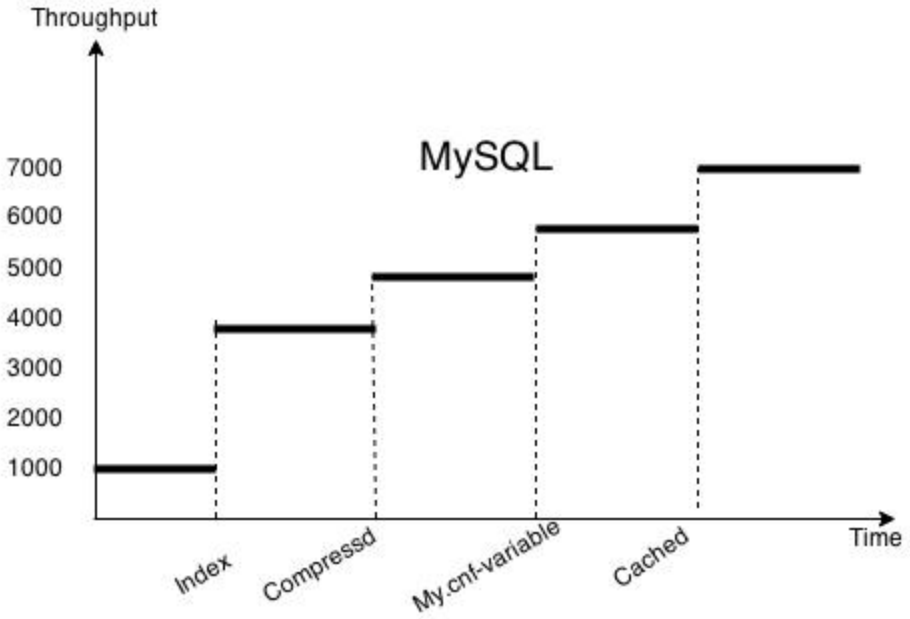
For Hbase, replication and sharding was done by itself automatically. All we had control over was the size until region splits and to presplit the regions so that we can load values to different region servers, so the traffic does not all go to one region server. Hbase has its own indexing system, it uses the rowkey (userid+time) as the index to quickly find where in the hfiles the queried data exists.
(EDIT)

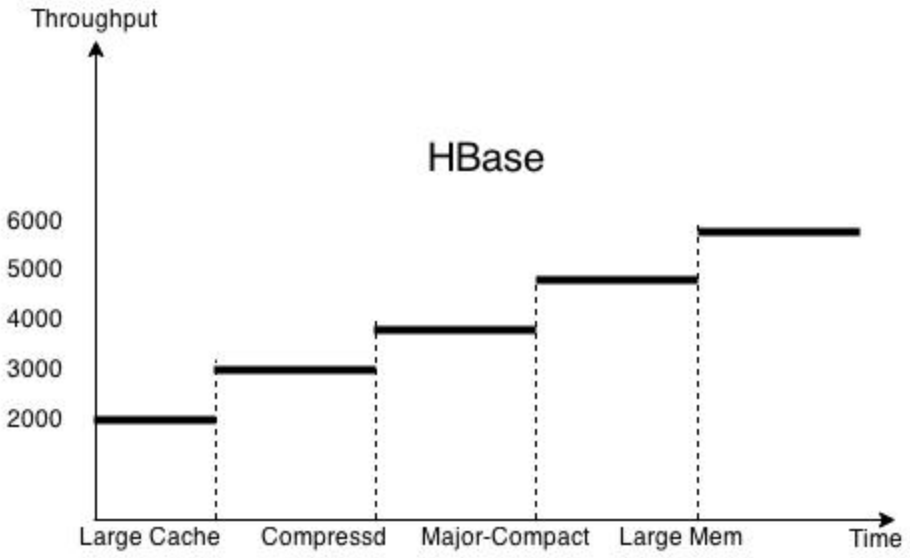
The other four optimization techniques were compression, multiple threading/connections, increasing memory (RAM) used, and decreasing IO wait time.

For mysql, since we used innodb and used almost all our time on hbase, we didn't feel the need to use compression for our data. As for multiple threading/connections, we sinply used threading at the front end to have multipled connections to the backend. We also changed multiple variable values that's provided in my.cnf.inno (in the hand in). The most important variable changed was key_buffer size, which allowed a faster search for indexes. Finally, for the decrease of IO wait time, we simply increased the used provision IOPS and thus there were no IOPS quota thus no wait time (%wa was around 0).

4. **Plot a graph showing results with/without each individual optimization that you used. Extremely impressive will be a timeline of rps v/s submission id (mentioning which optimization was in use at that time).**

We use the same plot charts given in question 2 of this part. It shows individual factors that impact on the final rps.





5. **Would your design work if your web service also implemented insert/update (PUT) requests? Why or why not?**

For Mysql, definitely not. This is because our database used myisam engine with compression. That automatically makes the table read only.
(EDIT) Since for phase 2, we changed our engine to innodb, thus we will be able to perform insert and update requests.

For hbase, the cluster will allow put requests. However, it would not be optimized with the

current configurations.

6. **Which API/driver did you use to connect to the backend? Why? What were the other alternatives that you tried? What changed from Phase 1?**

We used the jdbc driver for mysql. We didn't try other alternatives, and found jdbc to be sufficiently fast. http://jtds.sourceforge.net/benchTest.html

We used hbase client API for Hbase. We didn't try other alternatives because there are a lot of resources for references in the internet and the API works efficiently.

Our bottleneck, latency was never caused by parsing. This was because when doing map, we already parsed and loaded the desired results into the db. Thus we did not put too much of our time to investigating which driver was the fastest, instead we invested most of our time in optimizing IO wait time.

(EDIT) for hbase, one caveat was for Q4, in which we collated all the dates together as a single column family. So we have to use Hbase's API so extract the columns we want from a single column family without retrieving the entire column family.

7. **Can you quantify the speed differential (in terms of rps or Mbps) between reading from disk versus reading from memory? Did you attempt to maximize your usage of RAM to store your tables? How much (in % terms) of your memory could you use to respond to queries?**

Reading from memory is around 1 to 2 orders of magnitude faster than reading from disk. Reading from disk should be 300 to 20000 requests per second for SSD. On the other hand, if directly reading from memory, the rps should be at least 15000 and above. (restricted by bandwidth) For Mysql, we did not use RAM to store our tables at all. Instead we allocated most of our RAM for index searching (key buffer size), while for Hbase, we allocated around 5G of heap size per data server. Since our memory was 8G (RAM), just to be safe from swapping, we used 5G for heap for hbase and 5G for key buffer size for mysql. $\frac{5}{8}$ was around 62.5%.

8. **Does your usage of HBase maximize the utility of HDFS? How useful is it to have a distributed file system for this type of application/backend? Does it have any significant overhead?**

Our Hbase without a doubt didn't FULLY MAXIMIZE the utility of HDFS, however we did tune the size of HDFS to be smaller, which allowed a higher random read throughput. It's very useful to have a distributed file system for our application given if we know how to fine tune the system, since we don't have to vertically scale the storage, processing speed etc. In addition, we can replicate the data so it provides both fault tolerance and network partition tolerance. Of course there'll be a lot of overhead. Since HDFS has a huge layer of abstraction that is constantly doing work. For instance, Hfiles are abstractions of metadata and sequence files that only allow addition/appending data. There will be a lot of overhead mapping master to region server, to regions to HFiles then to sequence files.

9. **How can you reduce the time required to perform scan-reads in MySQL and HBase?**

Since we don't do much scan reads for any of our data, we didn't optimize scan-reads. However, if we were to do that, we can increase the memory used for sort_buffer_size, read_buffer_size, read_rnd_buffer_size and join_buffer_size. It allows sorting and scan reads to have more memory per connections. In addition, we can allow the read-ahead function that's built in mysql, so for a scan, it reduces the time done for reading.
As for hbase, we didn't optimize scan-reads as well, because all of our gets were performed as a single row get. However, if we were to increase the scan-read performance, we could do read aheads as well.
http://search-hadoop.com/m/qOo2yyHtCC1

10. **Did you use separate tables for Q2-Q4 on HBase and MySQL? How can you consolidate your tables to reduce memory usage?**

Yes, we use separate tables. We didn't consolidate our tables, however we had a even better idea of consolidating all the columns into a single column. Thus if we queried A, and we have tables that map A -> B, then we get B directly instead of getting more data from different columns and different rows. By creating an index on A, we can get B really quickly.

11. **How did you profile the backend? If not, why not? Given a typical request-response for each query (q2-q4) what <u>percentage</u> of the overall latency is due to:**
    a. Load Generator to Load Balancer (if any, else merge with b.)
    b. Load Balancer to Web Service
    c. Parsing request
    d. Web Service to DB
    e. At DB (execution)
    f. DB to Web Service
    g. Parsing DB response
    h. Web Service to LB
    i. LB to LG

    How did you measure this? A 9x6 (Q2H to Q4M) table is one possible representation.

	Q2H	Q3H	Q4H	Q2M	Q3M	Q4M
a	0%	0%	0%	<3%	<10%	<10%
b	0%	0%	0%	<3%	<10%	<10%

c	<1%	<5%	<1%	<1%	<5%	<5%
d	<1%	<5%	<1%	<3%	<10%	<10%
e	>90%	>50%	>90%	>60%	>30%	>30%
f	<2%	<5%	<2%	<3%	<5%	<5%
g	<1%	<5%	<1%	<1%	<10%	<10%
h	0%	0%	0%	<3%	<10%	<10%
i	0%	0%	0%	<3%	<10%	<10%

12. What was the cost to develop your back end system?

Almost $20

13. What were the best resources (online or otherwise) that you found. Answer for HBase, MySQL and any other relevant resources.

MYSQL:
http://www.soliantconsulting.com/blog/2012/09/mysql-optimization-faster-selects-myisam-fixed-row-format
http://www.toadworld.com/platforms/mysql/w/wiki/6178.tables-key-cache-tuning-of-myisam.aspx
http://www.techrepublic.com/article/save-disk-space-by-compressing-mysql-tables/
https://signalvnoise.com/posts/3571-scaling-your-database-via-innodb-table-compression
http://web.performancerasta.com/2-2-data-analytics-myisam-vs-archive-vs-infobright-ice-vs-infobright-iee-benchmark/
https://tools.percona.com/wizard
http://www.lemug.fr/wp-content/uploads/2011/01/Wp_MySQL-5.5_InnoDB-MyISAM.en_.pdf
http://doc.ctrlaltdel.ch/database/mysql/manual_Performance.html
http://www.messagepassing.com/2009/07/fast-compact-myisam-tables/
http://www.toadworld.com/platforms/mysql/w/wiki/6178.tables-key-cache-tuning-of-myisam.aspx
http://www.toadworld.com/platforms/mysql/w/wiki/6387.query-tuning-slow-query-log.aspx
http://www.databasejournal.com/features/mysql/article.php/3367871/Optimizing-the-mysqld-variables.htm
http://www.mysqlcalculator.com/

http://www.fromdual.com/mysql-performance-tuning-key#201

HBASE:
http://www.percona.com/blog/2009/08/06/why-you-dont-want-to-shard/
http://www.slideshare.net/vanuganti/hbase-hadoop-hbaseoperationspractices
http://blog.cloudera.com/blog/2012/08/hbase-replication-operational-overview/
http://www.slideshare.net/cloudera/hbasecon-2013-compaction-improvements-in-apache-hbase
http://www-01.ibm.com/support/knowledgecenter/SSPT3X_3.0.0/com.ibm.swg.im.infosphere.biginsights.analyze.doc/doc/bigsql_compaction.html
http://www.ngdata.com/visualizing-hbase-flushes-and-compactions/
http://www.larsgeorge.com/2009/10/hbase-architecture-101-storage.html
http://abloz.com/hbase/book.html#mapreduce

(EDIT) For phase 2

http://cassandratraining.blogspot.com/2013/05/apache-hbase-default-configuration.html
http://www.oracle.com/technetwork/java/javase/tech/g1-intro-jsp-135488.html
http://www.oracle.com/technetwork/java/javase/tech/g1-intro-jsp-135488.html
http://hortonworks.com/blog/hbase-blockcache-101/
http://www.ericsson.com/research-blog/data-knowledge/hbase-performance-tuners/
http://ronaldbradford.com/blog/bigint-v-int-is-there-a-big-deal-2008-07-18/
http://ronaldbradford.com/blog/bigint-v-int-is-there-a-big-deal-2008-07-18/
http://planet.mysql.com/entry/?id=13825
http://search-hadoop.com/m/qOo2yyHtCC1
http://blog.cloudera.com/blog/2012/06/hbase-io-hfile-input-output/

google’s bigtable paper : BigTable: A Distributed Storage System for Structured Data

[Please submit the code for the backend in your ZIP file]

Task 3: ETL

1. For each query, write about:
    a. **The programming model used for the ETL job and justification**

For query 3 and query 4, we both used Elastic Mapreduce on Amazon as the programming model for the data extraction and transforming job. We used 1 master and 18 cores of m1.large to these jobs. We have 1.2 Tb compressed raw data from S3. It will take a very long time if we use only one instance to do the ETL job. EMR can effectively extract data from s3 bucket with compressed files as input stream. And it can apply mapreduce jobs on hadoop cluster, which can decrease the whole time cost to process data parallelly.

a. The number and type of instances used and justification

For both q3 and q4, we used 1 master and 18 cores of m1.large to do the final extraction and transforming jobs. More instances can improve efficiency of data processing and large instance can also be more helpful than small and medium. m1.

b. The spot cost for all instances used

For ETL in the whole phase 2, spot cost for all EC2 instance are: 5.54(jiali) + 10.50(ziyuan) = 16.04

c. The execution time for the entire ETL process

	Extraction and Trasformation(MapReduce)	Load(mysql)	Load(Hbase)
Q3	1 hour, 32 minutes	5 min	1hr
Q4	1 hour, 29 minutes	7 min	24hr

d. The overall cost of the ETL process

For total q3 and q4:

EC2: 5.54(jiali) + 7.50(ziyuan) = 13.04

Elastic MapReduce: $3.34(jiali) + 15.57(ziyuan) = 18.91

Total: 8.88(jiali) + 23.07(ziyuan) = 31.95

e. The number of incomplete ETL runs before your final run

6 (incomplete + test try) for phase 2

f. Discuss difficulties encountered

For q3, we got a too complex idea in extracting and counting relations between userid and it's reteet userid, after brainstorming in team, we got a better idea in dealing with the problem. We record + and - relations in mapper program, and using word counting method and hashmap data structure in reduce to count their relations and get * relations.

For q4, we got stucked in the idea of how to store data in database. For mysql, we decided to store the data in 3 columns. hashtag, tweetdate, tweetid+userid+tweetTime. For Hbase,

we set rowkey as hashtag, set unique tweetdate as different column qualifier in the same family. Since we misperceived the maximum date distance, which is actually about 700+ days, we failed in ETL mapreduce 5 times. And finally we had to load data using put commands, which cost us about 24 hours, but fortunately it loaded successfully.

Another problem we met in both q3 and q4 is that we ignored deleting duplicated records, which result in some errors in the live test. We tried to solve it, but went in wrong direction again. We changed the key to userid_tweetid as the output of mapper, and delete the same key in reduce program. It would work well if the sort function in mapreduce is in alphabetical order, however it is not, it just simply put the same key in the same reduce. So next time, we will append tweetid at the end of value rather than key position to solve the duplicated problem.

1) **The size of the resulting database and reasoning**
   q3: 3.1G
   q4: 4G

   g. The size of the backup
   For mysql and hbase, we choose 100GB EBS to back up table data of q3 and q4.

2. What are the most effective ways to speed up ETL? How did you optimize writing to MySQL and HBase? Did you make any changes to your tables after writing the data? How long does each load take?

USE EMR to ETL . It can do ETL fast and save it on s3 directly. We use EMR to produce the data as the format we want. We compress, index, compact, keep in-memory of hot columns or rows of tables. For mysql, each load will be fewer than 15 minutes. For hbase, if we use bulkload, it will cost fewer than 15 minutes.

3. Did you use EMR? Streaming or non-streaming? Which approach would be faster and why? How can you parallelize loading data into MySQL?

We always use streaming. EMR is too expensive, there is no too many chances to try EMR without experience.
No, we do not parallelize loading data. We only use Load sql statement in MySQL

4. Did you use an external tool to load the data? Which one? Why? If you were to do Q2 (Phase 1) ETL again, what would you do differently?

We always use the same way to load data for MySQL. For HBase, we change to use java api to load data, because we need to load much flexible data. That mean, a bulk loading can meet our needs. We did not try our interface tools to load. If there is time or opportunity, I will try some tools like hive.

5. Which database was easier to load (MySQL or HBase)? Why?

I think MySQL is easier. There is a fixed schema for creating, setting and loading. But, for HBase, there is no schema. So we just use HBase shell to create and setting, and use HBase api to load. When loading data, we need to put local data to hadoop. Then we need to bulkload and finally adding to a table. This process is a little complex and easy to make mistakes.

[Please submit the code for the ETL job in your ZIP file]

General Questions

1. Would your design work as well if the quantity of data would double? What if it was 10 times larger? Why or why not?

Yes, we will change the design. From the experience of phase 2, the q2 has every difficult to have a high score both for mysql and hbase. So when the quantity is larger as 10 times, I think I will change the way to access data; however, if it is only doubled, we will most likely not change it.

2. Did you attempt to generate load on your own? If yes, how? And why?

No. I never attempt to generate load on my own. You give us a generator, why do we need to create a new one? If it is for a warmup for ELB, I think it is useful to a quick warmup. However, it is fairly easy to generate load on our own with a few queries, but it'll be harder to generate random queries that's evenly across multiple keys. In addition, with the 60 dollar limit, we needed to conserve money for the more important purposes.

3. Describe an alternative design to your system that you wish you had time to try.

For hbase, we use a cluster for all tables. But I think if for a mix testing, it will be better to use separate clusters. From this guess, I will try if it is possible. I still understand why my q2 and q3 have such different throughput, just because of a large file? If it is, I think split regions more than 1000 to try if it will have better performance.
In addition, we would try out https://github.com/sematext/HBaseWD, which allows us to manually hash our keys. In addition, we would try out different sizes.

For mysql, we think we can try a standalone server and only one frontend with 20000 IPOs. Thus we will be able to get a full score for Q1 and not have such a complicated infrastructure. In addition, I will try to compress all tables and try to limit connections and threads.

If you can read here, I actually have a question, you have a live test with mixing queries. Why there is no test try on your system. It will not allow us to know how our system is in performance.

4. Which was/were the toughest roadblock(s) faced in Phase 2?

No one to ask. For just a stupid question, we need to figure it out for more than 5 hours. When I post it to piazza, the only answer is "do you search it on google?" or "you can try this file". Even after phase 1, I do not know how to improve performance of the system systemically. I just have to try and stupidly laugh for a high score that may be due to a network fluctuation. We know all teams are working like this, but I do not think it is a good way to learn.
In addition, for hbase, we found it is difficult to find the detailed performance for each testing. It is really hard to evaluate this tuning. I mean there is fewer tools for us to debug. It is a little hard for us to understand how to tune the hbase system, like the relationship between hadoop and hbase, which parameter may have the largest influence on the

throughput.
Finally, for mysql, there is less difficulty to do it.

5. Did you do something unique (any cool optimization/trick/hack) that you would like to share with the class?

We used a trick called sleep_time=0. It greatly improved our throughput.

[NOTE:

- **IN YOUR SUBMITTED ZIP FILE, PLEASE DO NOT PUT MORE ZIP OR GZ FILES (NO NESTED ARCHIVES)**
- **USE A SIMPLE FORMAT**
  - **/etl**
  - **/frontend**
  - **/backend**

]