S15 15619 Project Phase 1 Report

Performance Data and Configurations

Best Configuration and Results	
Number and type of instances	Q1: 4 m1.large instances+elb Q2H: 6 m1.large instances Q2M:3m1.large+ebs_optimized,3m3.large+elb
Cost per hour	Q1: 1elb+4m1.large=0.025+4*0.175=0.725 Q2H: 6*0.175 = 1.05 Q2M:0.175*3+0.025*3+0.140*3+0.025 = 1.045
Queries Per Second (QPS)	INSERT HERE: (Q1,Q2H,Q2M) score[20, 109.01 134.29] tput [18434.0, 5995.8 7386.0] latcy [5, 8 6] corr [100.00, 100.00 100.00] error [0.00, 0.00 0.00]
Rank on the scoreboard:	Q1: 3 Q2H: 10 Q2M: 10 phase1: 6

Team : Oak

Members :Ziyuan Song, Aaron Hsu, Jiali Chen

Rubric:

Each unanswered question = -5%

Each unsatisfactory answer = -2%

[Please provide an insightful, data-driven, colorful, chart/table-filled, humorous and interesting final report. This is worth a quarter of the grade for Phase 1. Use the report as a record of your progress, and then condense it before sharing it with us. Questions ending with "Why?" need evidence (not just logic)]

Task 1: Front end
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. In addition, it allowed a pipelining, as we

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

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.

3. **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.
(EDIT) During Q2, 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).

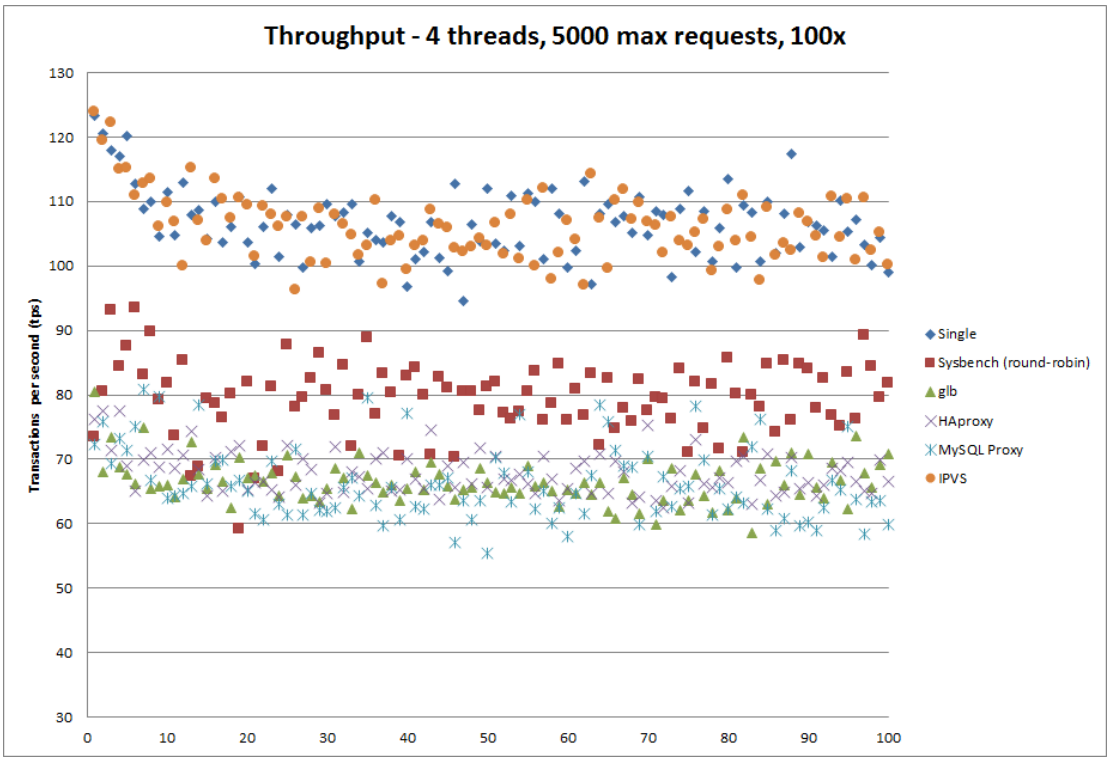
4. **Did you use an ELB for the front-end? Why, or why not? Condense your experience with ELB in the next few sentences. Talk about load-balancing in general and why it matters in the cloud.**

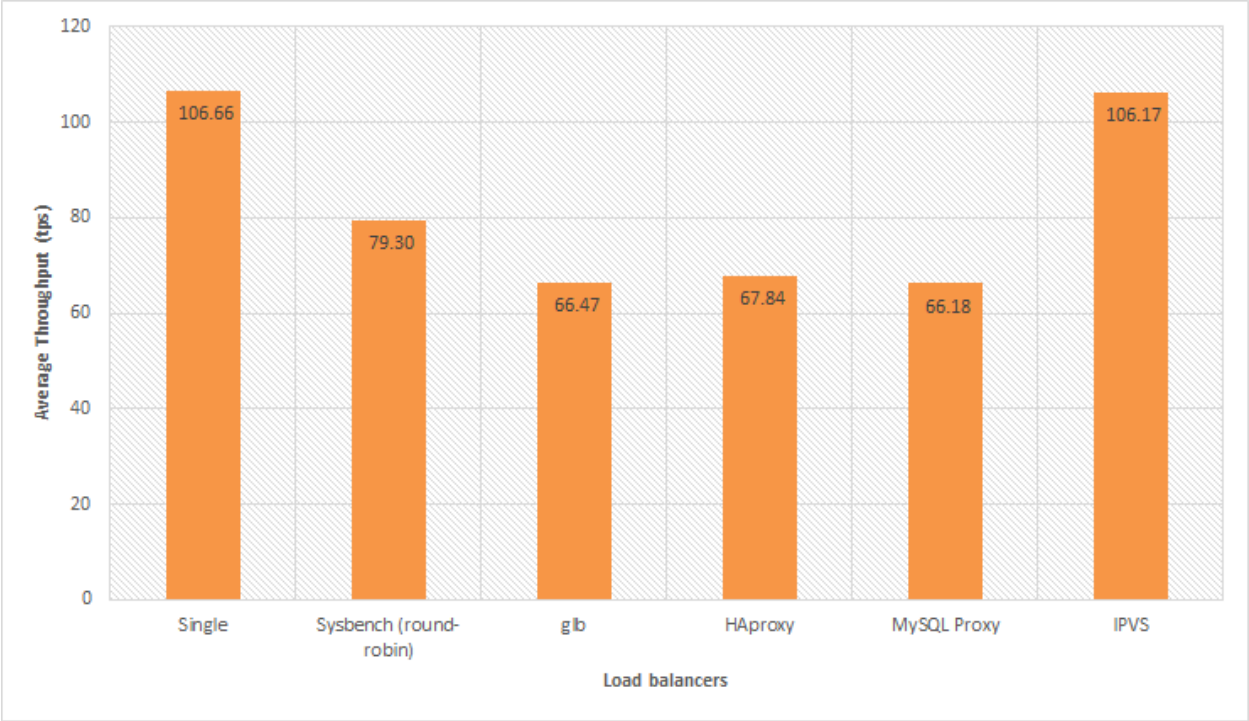
Yes, we used an ELB for the front-end. Because ELB can automatically route traffic across multiple instances to achieve higher levels of fault tolerance, it also automatically scales its request handling capacity to meet the demands of application traffic, since we had multiple front ends for achieving high RPS. It reduces the bottleneck by spreading the traffic to multiple front ends, routing to multiple backends.

5. **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.**

Yes, 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.)

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, reaching 5500 before investing more time for the complicated IPVS switcher, we decided to test out Haproxy first before the deadline for phase 1. We will most likely test out IPVS in phase 2.



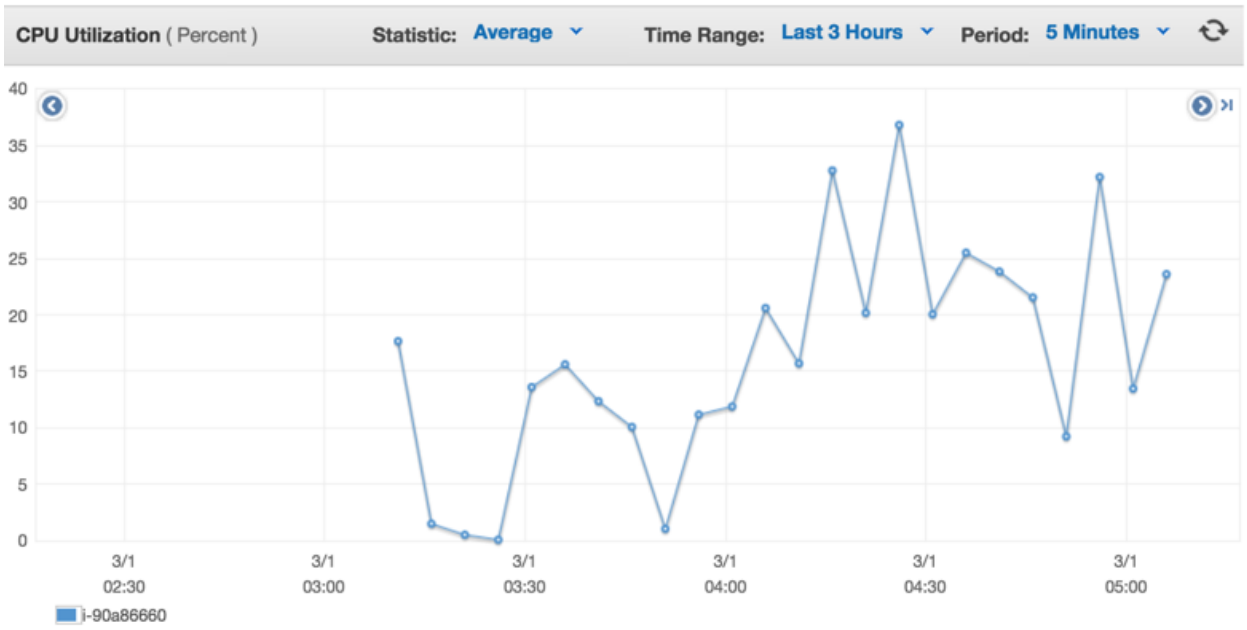


6. **Did you automate your front-end instance? If yes, how? If no, why not?**

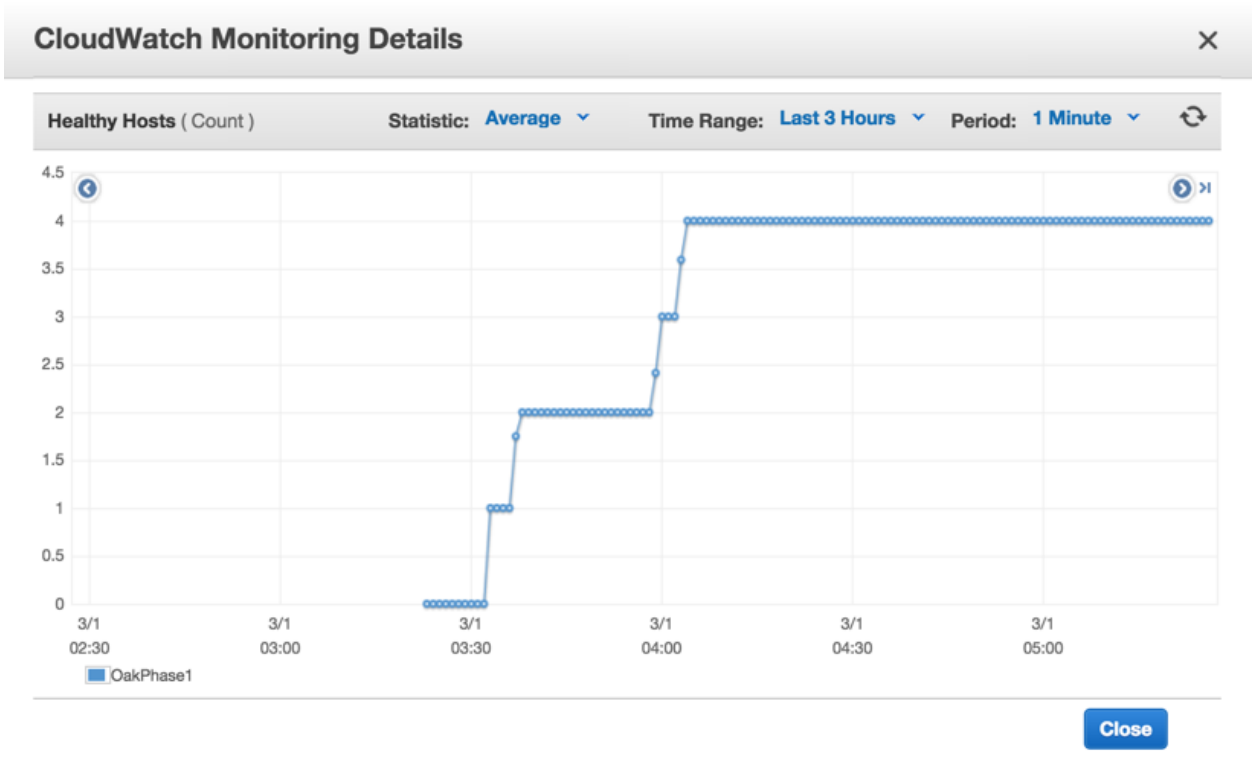
Yes, we wrote a shell script that updates, installs all required packages and added the script that initiates the front end server. Then we added the command that initiates the front end server to crontab, specifying that the server should be initiated when system startup. Then, we created an amazon 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?).

7. **Did you use any form of monitoring on your front-end? Why or why not? If you did, show us the results.**

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.



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

17.54+2.85+0.748 = 21.138

9. **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. 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.

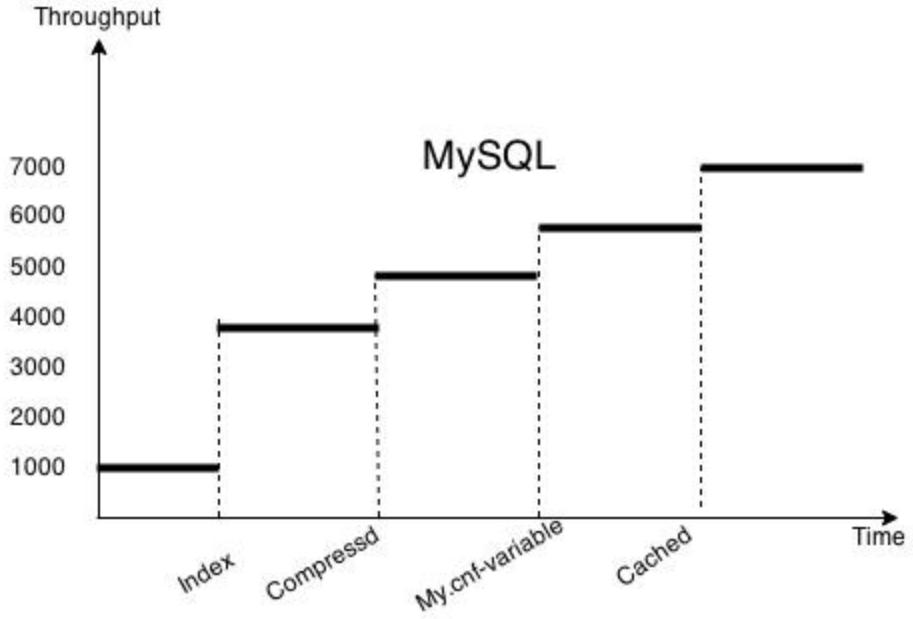
For **mysql**, our first column (userid+time) was not unique. At first, we were hesitant about which engine to use, innoDB or Myisam. However, after closer evaluation, we chose to use Myisam because Myisam has several optimization options that specializes for reading operations. The first option was 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

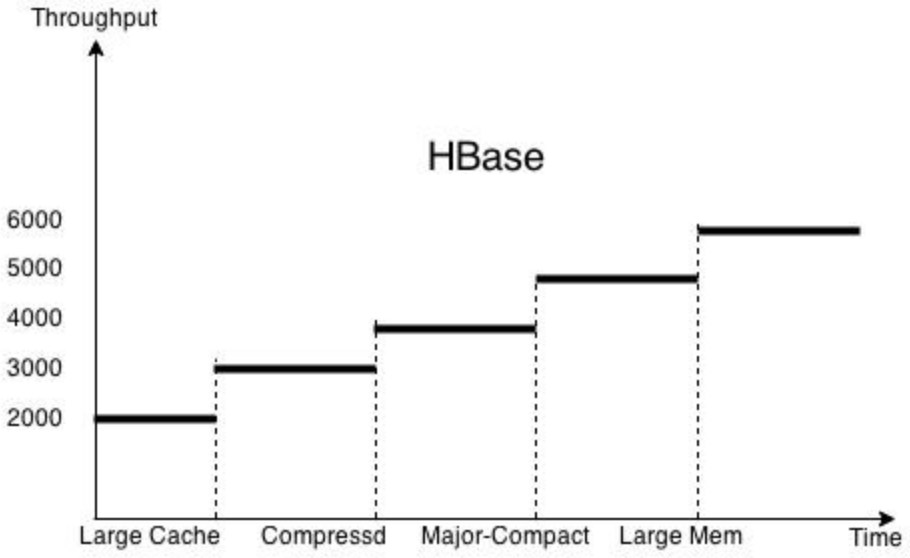
2. **What was the most expensive operation / biggest problem with your DB that**

you had to resolve for Q2? 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.



3. **Explain (briefly) <u>the</u> <u>theory</u> behind (at least) 3 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).**

    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.

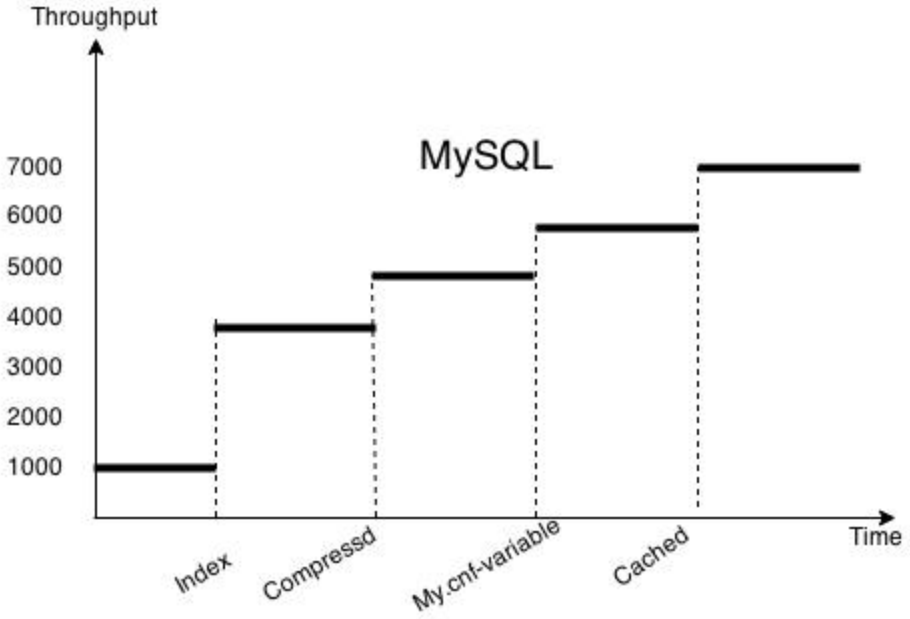
For MySQL, replication and sharding is done via a mysql cluster. An API node would be the communication node between front ends and all its data nodes, while a master node allows configuration of the mysql cluster. A user could specify the number of sharding or replication nodes through the master node.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

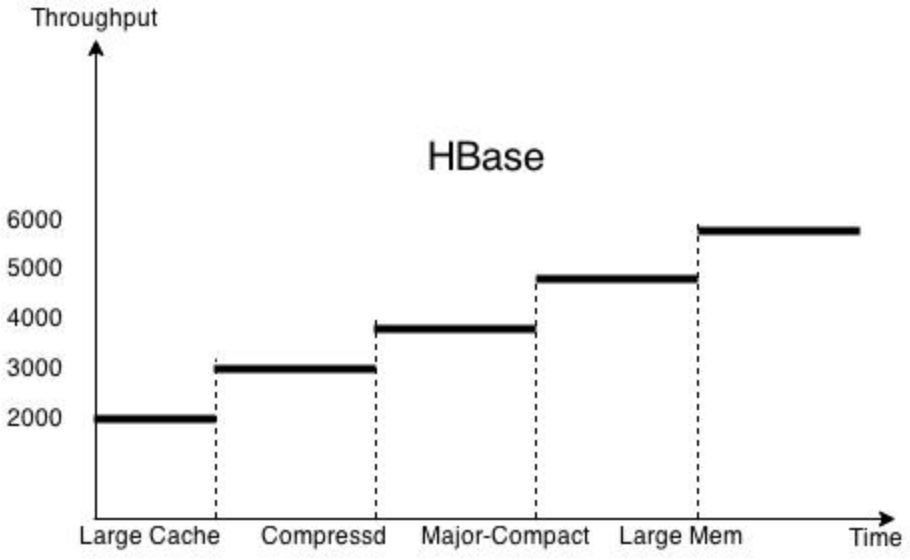
For Hbase, replication and sharding was done by itself automatically. All we had control over was the size until region splits and to presplit the regions so that we can load values to different region servers, so the traffic does not all go to one region server. Hbase has its own indexing system, it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

the hfiles the queried data exists.

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.

On the other hand, 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?**

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.

7. **How did you profile the backend? If not, why not? Given a typical request-response for each query (q1-q2) what percentage 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 9x2 table is one possible representation.

In order to monitor our backend, we used iostat, sysstat, top, free.. etc to monitor the %id (available CPU), %wa (IO wait time),free to use RAM, and to see whether there's swapping occurring.
First, if there's a huge %wa, that means the bottleneck is at the backend IO read time. If there's a large CPU usage, it means we're not allocating enough memory or cache. We also had to ensure there was no swapping occurring caused by too much memory/cache allocation for the backend, or else it would further slow down the backend.

Q1:
First row: mysql
second row: hbase

0.222	0.222	0.111	0	0	0	0.222	0.222
0.222	0.222	0.222	0	0	0	0.222	0.222

Q2:
First row: mysql
second row: hbase

0.1	0.1	0.05	0.1	0.4	0.05	0.1	0.1
0.078	0.078	0.028	0.078	0.6	0.028	0.078	0.078

8. **Say you are at any big tech company (Google/Facebook/Twitter/Amazon etc.). List one concrete example of an application/query where they should be using NoSQL versus one where they should be using an RDBMS. Both examples should be based on the same company (you choose).**

I will use google as an example. Web searching engine should use NoSQL database. It hbase a different data model, and its amount of data is tremendous. NoSQL (Big Table) is better for flexibly writing with new models. And it do not have to follow ACID which will decrease the whole performance of querying. In addition, NoSQL supports sharding naturally, while RDBMS have to make more configuration that cannot be finished by Mapreduce easily. So such data storage should apply NoSQL database. On the other hand, the traditional RDBMS is better for more strict relational data, like user-login system. It follows ACID that can guarantee data consistency and security performance.

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

Total: 23.3

10. **What were the best resources (online or otherwise) that you found. Answer for both HBase and MySQL.**

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

[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**

Extraction: We used Elastic Mapreduce on Amazon as the programming model for the data extraction job. We used 1 master and 18 cores of m1.large to these jobs. We have 1Tb 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.

Transmission: We use s3cmd tool to get data from S3 to local EBS.

Load:

For mysql, we used load in file, which is more efficient. We also indexed after completely inserting, or else it will sort the index after every insert, which will be immensely slow.

For hbase, we use importTsv and bulk load tool to load data to installed hbase on hadoop.

    b. **The number and type of instances used and justification**

We use 1 master and 18 cores of m1.large to do the final extraction jobs. More instances can improve efficiency of data processing and large instance can also be more helpful than small and medium. m1. For **mysql**, we use 3 instances of m1.large as databases, because m1.large has a better performance for IO with optimized EBS. we use other 3 instances of m3.large as front-end server due to its more cores on each one. For **hbase,** we use one large master and 4 cores that make up an EMR cluster. We try the standalone mode of hbase and one backend can have a relative good performance, but it may be not good for later live testing. So we decide to use a hbase cluster

    c. **The spot cost for all instances used**

EMR cluster instances: 0.016*19*2hr = 0.78
One Mysql instance: 0.016*1*1hr = 0.016
One Hbase cluster: 0.016*5*1hr = 0.08
Total : 0.876/per time ETL.

    d. **The execution time for the entire ETL process**

Extraction : 1h40m
Transmission : 30m
Load: 1hr(mysql), 1hr(hbase)
Total: 250m

    e. **The overall cost of the ETL process**

14.95(ziyuans)+0.44(jialic)+0.48(Aaron) = 15.87

    f. **The number of incomplete ETL runs before your final run**

15 (incomplete + test try)

    g. **Discuss difficulties encountered**

1) Extraction: When we use EMR to extract data, we originally use version 2.4.2 hadoop. We always find, there is special error characters output from reducers. The reason

why that happened is we do not use escaped text as input stream. It cost us a whole day to figure that out. Then we became to use a high version of hadoop. The output works fine.

2) Load: we try to load utf-8 data to mysql database, but we find there are lots of special characters in the text, like '\n', '\t' and so forth. Because there is '\n' at the end of each text, it will be hard to load data by line and separate data by '\t'. Then we decide re-do the extraction process and insert some special characters to mark separates fields and lines.
3) encoding: When we load the data, using only UTF-8 as the encoding method is not enough, some arab characters can not shown correctly in UTF-8. So we use UTF8mb4 as the database charset instead.

h. The size of the resulting database and reasoning

For mysql, the original size is 47GB. After compressing, it becomes 29GB
For hbase, the original size is 90GB. After compression, it becomes about 40GB

i. The size of the backup

For mysql, we choose 100GB EBS to back up table data.
For hbase, we choose S3 to back up compressed 40GB data.

2. What are the most effective ways to speed up ETL?

The first way is to use EMR with more core instances to do extraction jobs. And for hbase, using the load bulk tool will decrease time to load data to distributed hbase. For each data load, it is better to back up last loaded data, it will save repeated data loading.

3. Did you use EMR? Streaming or non-streaming? Which approach would be faster and why?

We only use streaming EMR. Because it can work well, we did not consider to use other methods. But I guess, streaming will be faster. Because a pipe-and filter structure can allow data processing asynchronously. Quoted from the paper below, "Their system generates MR outputs by reading the items within the window. This stream-based MapReduce processes arriving increments of update tuples, avoiding recomputation of all the tuples from the beginning." http://www.cs.arizona.edu/people/bkmoon/papers/sigmodrec11.pdf

4. Did you use an external tool to load the data? Which one? Why?

No, we do not use an external tool. We only used hadoop's own bulk load tool. However at the end, we found out that the queries from the load generator ask for queries with incrementing userid. Since we did not use any hashing function to load the data to different region servers, all the work was directed to a single region server. So in phase 2 we are planning to use https://github.com/sematext/HBaseWD so we can balance the work load across different region servers.

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

MySQL is easier. For mysql, it was simply making a definition of a table, then loading the

data directly to a volume. When we load data to HBase, we used Hadoop platform. We need to put input data to HDFS and turn it into Hfile. After that, we made it bulk load so the loading process is faster. Bulk loading will make things more complicated, but more efficiently.

[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?**

For this phase, we do think we need to change the design, because the generated data have particular pattern. For MySQL, we use duplications databases to handle more requests, and for HBase, we use pure sharding to do that. If data is doubled or ten times, only a particular block of data is requested, which will not influence on our system. However, if the request pattern changed in the future, like random requesting, we my mix duplicating and sharding methods in two databases to improve the whole efficiency of usage. For us, the data changing from 1TB to 10TB is not a determining factor to design our system, while the way how to query data from the generator is the main factor.

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

No, we have not do that. But this question indeed hints us to do that for the later project. We find it is hard to warm up ELB. When it is eventually warmed up, we have to wait for more 100 seconds to run the next programs. So we may have to run a script to generate more traffic during the warming up period. It should be efficient to warm up system.

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

For front-end, we have not designed any cache to response requests. Front-end caching is a cheap way to improve performance; however, we only focused on back-end optimization. I do we may have a random or a different way for a future live testing, so a good algorithm for caching in front-end will be more helpful. But I think a pure LRU way to cache is not good enough to handle a large traffic, so we need to try to optimize it in the next week.
For hbase, we find there is a “hot-region” problem. There are so many queries on a same region or region-server. So we guess if we split the table with a special way to separate data into different regions or servers, the whole performance on this cluster will be improved. We might try HbaseWD solution next week.
For MySQL, we have tried to do all kinds of ways to optimize back-end, like compression, optimized-ESB, no locks with MyISAM, large caching, removing logs, and so forth. Sometimes, we can have a very high performance, but we still cannot keep it on a stably status all the time. So next step, we should consider how to keep it on the same performance level all the time.

4. **Which was/were the toughest roadblock(s) faced in Phase 1?**

It is very hard to debug in Phase 1. We have fewer tools to find out our bottleneck if we have some problems. It forced us to do lots of assumptions or guesses on why it is not stable, how to explain various cases, and how to adjust performance parameters.

In addition, ETL costs too much time if we have some faults in the data after processing. Some repeated and waiting time kept us a low morale for a rather long time.
Finally, when we work on both databases, we tried to optimize back-end first, and ignore front-end, which cost too much time to reach a high score.

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

At the end of the project, we found out the easiest hack was to do a 10 minute test that loads up the front end cache, for multiple front ends then make a ELB that connects to all the front end instances. Then simply perform multiple 1 minute tests and hope that all the queries hit the front end cache, resulting in the maximum throughput and minimum latency. (That was not what we did,we didn't even have a front end cache.)