Spring 16 15619 Project Phase 2 Report

Performance Data and Configurations

<table>
<tr><th colspan="2">Best Configuration and Results from the Live Test</th></tr>
<tr><td>Number and type of instances</td><td>HBase Live Test: 1 m3.medium, 6 m4.large
MySQL Live Test: 6 m4.large, 1 elb</td></tr>
<tr><td>Cost per hour
(assume on-demand prices)</td><td>HBase Live Test: $0.787
MySQL Live Test: $0.745</td></tr>
<tr><td>Queries Per Second (QPS)</td><td>INSERT HERE: (Q1,Q2H,Q2M,Q3H,Q3M)
<table>
<tr><th></th><th>Q1</th><th>Q2H</th><th>Q2M</th><th>Q3H</th><th>Q3M</th></tr>
<tr><td>score</td><td>6.25</td><td>11.0675</td><td>12.5</td><td>4.79</td><td>12.5</td></tr>
<tr><td>tput</td><td>32989.9</td><td>8887.3</td><td>27979.7</td><td>2299.2</td><td>12156.5</td></tr>
<tr><td>ltcy</td><td>2</td><td>5</td><td>1</td><td>21</td><td>4</td></tr>
<tr><td>corr</td><td>100</td><td>100</td><td>100</td><td>100</td><td>100</td></tr>
<tr><td>err</td><td>0</td><td>0.38</td><td>0.44</td><td>0</td><td>0</td></tr>
</table>
</td></tr>
<tr><td>Rank on the scoreboard:</td><td>Q1(Higher One): 21
Q2H: 24
Q2M: 1
Q3H: 11
Q3M: 3
HBase Live Test: 9
MySQL Live Test: 2</td></tr>
</table>

Team : elder
Members :
Jialing Liu (jialingl)
Lei Jin (ljin1)
Hao Wang (haow2)

Rubric:

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

[Please provide an insightful, data-driven, colorful, chart/table-filled, <u>and interesting</u> final report. This is worth a quarter of the grade for Phase 2. 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 (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 use undertow as our front end framework.

Undertow is very good at handling HTTP requests and have great performance for it can create threads automatically, have very small package and easy to configurate.

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

We used both undertow and vertx in Phase 1 for comparison. Undertow has properties described above. Vertx also has a very great performance and it has its own thread pool to handle concurrent requests so we needn't to handle them by ourselves. It can pass distributed messages in a distributed way. It supports multiple programming languages and have a very small package and easy to do configuration.

But after comparing these two frameworks. We choose to use undertow since it's faster, lighter and could easily be compiled in gradle and maven project. We only need to add dependencies in maven's POM file or in build.gradle to compile for new jars. However, for vertx, we need to search the internet for appropriate jars and add them in library. So, undertow is more convenient for this project.

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

Total hourly budget $0.85 taken into account, we could have at most 6 m4.large or 5 m3.large.

After experiments, we found that the former could lead to better performance of our application. So we choose 6 m4.large to form our HBase cluster. And another m3.medium is used as cloudera manager so that we could configure HBase without taking too much master's memory. And we put the front end server on the master of the HBase cluster.

Similarly, we choose 6 m4.large as frontend servers for MySQL and use an ELB to load balance.

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

We changed the java heap space to 2G for cache and ram for java programmes. And we use an ELB for the front-end for MySQL system.

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 ELB as front-end to automatically distribute requests to 6 back-end MySQL servers with data replication. ELB is great to balance load, which performs much better than the load-balancer implemented by ourselves. Load balancer can better make use of servers. Besides, it can also check the health of servers so as to launch a new server when necessary. If use Round-robin, it means simply allocate the requests to a server, and it can also consider other features such as CPU utilization etc.

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.

We have also tried a load balancer implemented by ourselves, but it performs much worse than ELB, for ELB has many internal optimizations that our load balancer does not have.

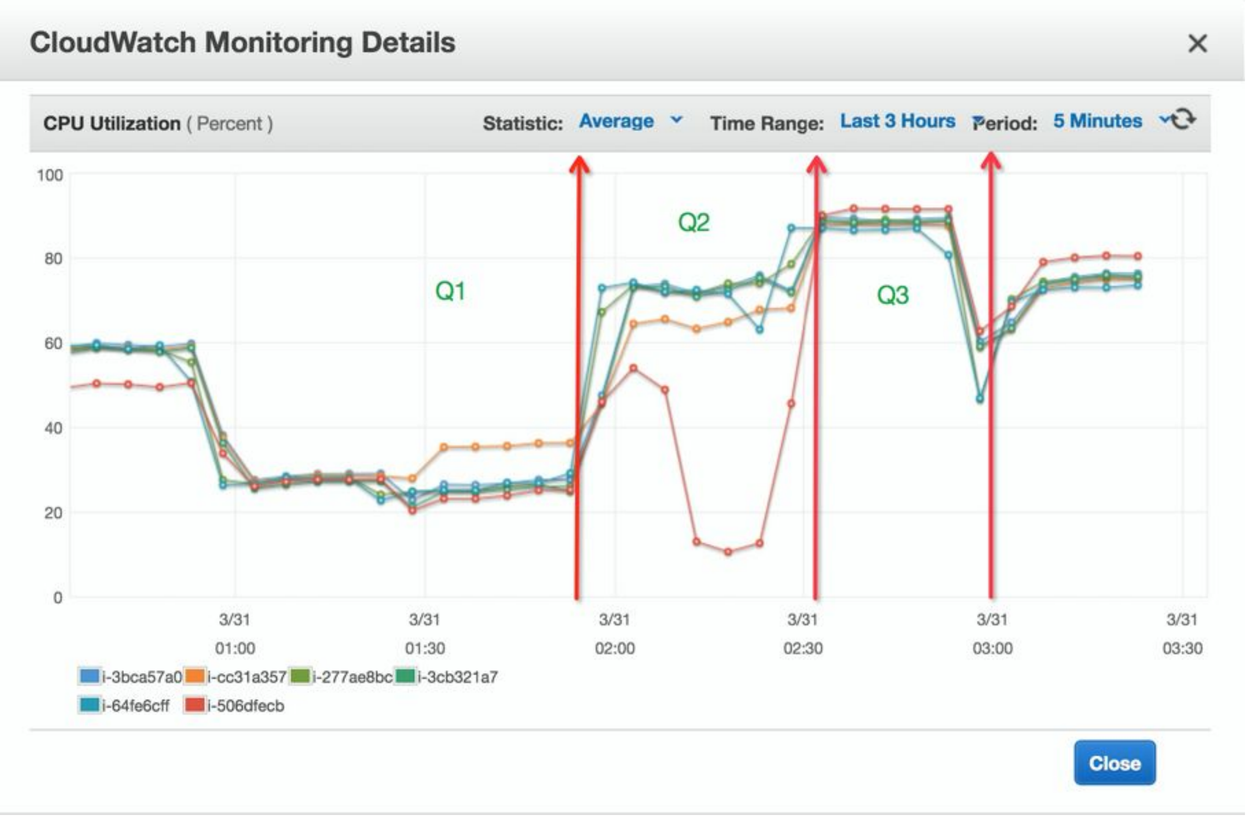
7. Did you automate your front-end instance? If yes, how? If no, why not?

Yes. We first construct all of configuration and codes into one instance, then we build an ami image of that instance. By creating several new instance using that ami, we can easily get the exact copies of the instances and use ELB to do load balancer.

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. Cloudwatch is very convenient for the visualization of requests got and sent, CPU utilization, instances health condition.

MySQL:
CPU Utilization:

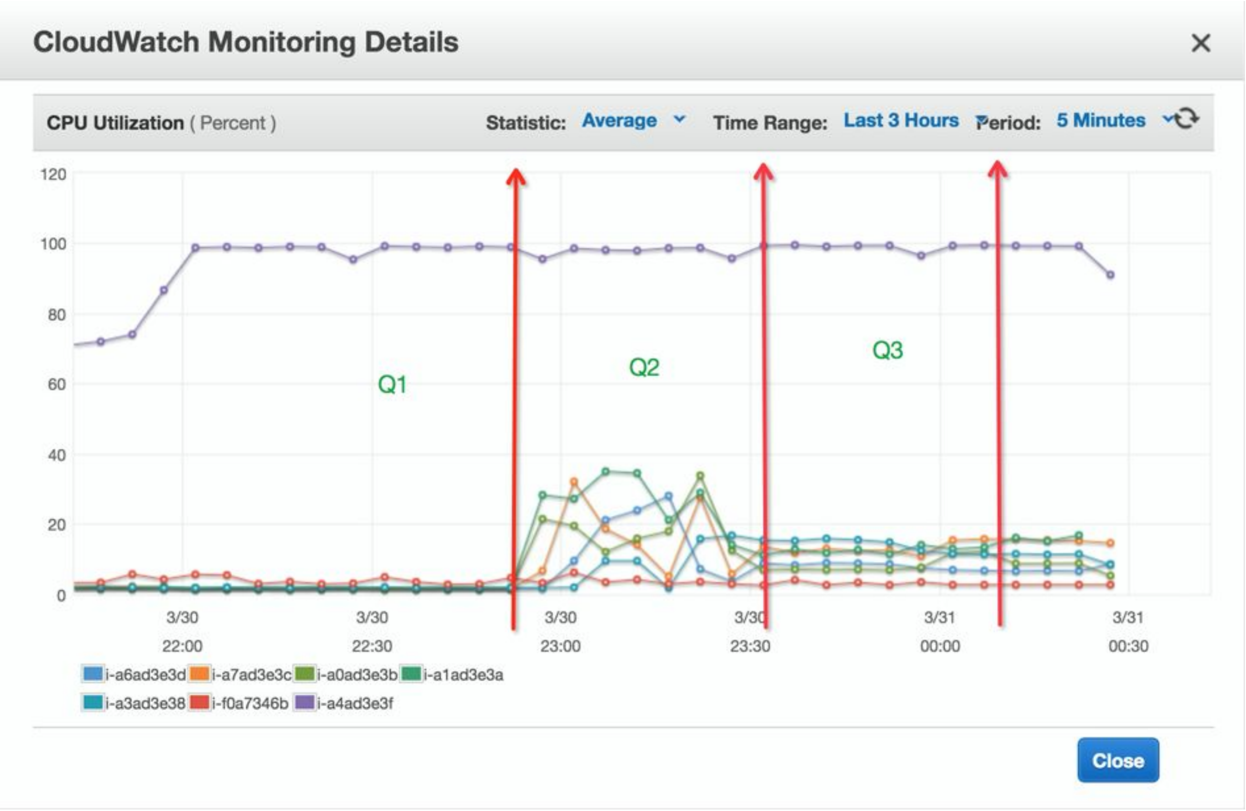


CloudWatch Monitoring Details ×

CPU Utilization (Percent) Statistic: Average Time Range: Last 3 Hours Period: 5 Minutes

HBase:

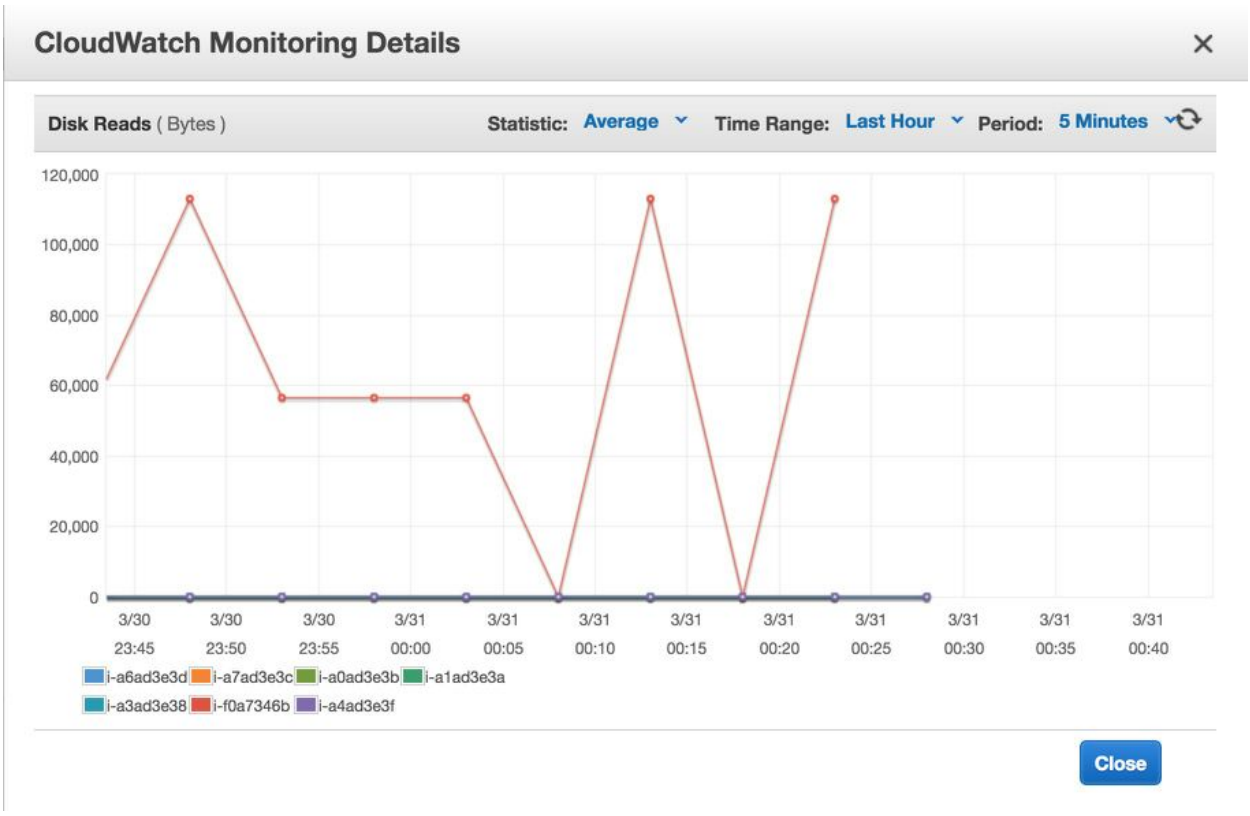
CPU Utilization:



CloudWatch Monitoring Details ×

CPU Utilization (Percent) Statistic: Average Time Range: Last 3 Hours Period: 5 Minutes

Disk Reads:



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

We spot or on demand a m4.large to write and debug our code and use about $15 for whole Q1 Q2 and Q3 session.

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

http://undertow.io/undertow-docs/undertow-docs-1.3.0/index.html
https://docs.gradle.org/current/javadoc/
http://www.cloudera.com/documentation/enterprise/5-5-x/topics/cm_intro_api.html
http://stackoverflow.com/

[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.

Q2:

MySQL schema:

PRIMARY KEY: userid_hashtag: varchar
allText: TEXT

Reason: By using userid_hashtag as the primary key, allText could be selected during very short time period.

HBase schema:

Row key	Column family
userid_hashtag: BINARY	allText: BINARY

Reason: By using userid_hashtag as the rowkey, allText could be gotten during very short time period.

Iterations:
MySQL: 54974923
HBase: 17156403

Q3:

MySQL schema:

userid: int unsigned
ymddate: int unsigned
words: TEXT

Reason: By indexing on userid and ymddate, we could do fast range query.

HBase schema:

Row key	Column family (data)
Userid: BINARY	info: BINARY

Reason: By using userid as the rowkey, we can do range scan on the table, i.e., setStartRow and setStopRow.

Iterations:
MySQL: 37885392
HBase: 24864713

2. Explain your choice of instance type and numbers of your back end system.

Total hourly budget $0.85 taken into account, we could have at most 6 m4.large or 5 m3.large.

After experiments, we found that the former could lead to better performance of our application. So we choose 6 m4.large to form our HBase cluster. And another m3.medium is used as cloudera manager so that we could configure HBase without taking too much master's memory. And we put the back end system on the HBase cluster.

Similarly, we choose 6 m4.large as back end system for MySQL.

3. Explain any non-default configuration and parameters you choose to set for your database system.

HBase:
Q2:

```
DATA_BLOCK_ENCODING => 'FAST_DIFF',
COMPRESSION => 'SNAPPY',
 BLOCKSIZE => '131072',
IN_MEMORY => 'true',
BLOCKCACHE => 'true'
```

Q3:

```
DATA_BLOCK_ENCODING => 'FAST_DIFF',
COMPRESSION => 'SNAPPY',
IN_MEMORY => 'true',
BLOCKCACHE => 'true'
```

MySQL:

```
key_buffer = 32M
max_connections  = 50
table_cache = 128
query_cache_type = 1
query_cache_size = 2310720M
```

4. 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.

MySQL: Concurrency. We first used single thread to do query, but it is extremely slow. Then we created multi-thread by ourselves, but it is still very slow. Finally we used connection pool and it is much faster than before.

HBase: At first, we use word_date as the row key. However, the size of respected column and the size of the whole table had bad influence on query speed. Since we need to do range scan, it will probably return us a very long text. So the size of the column negatively affects speed. Firstly, we split the column by using mod function on userid and query specific rows. However, this trial failed because get a lot of lines from a pretty large table is slow. Secondly, we change our schema and use userid as the row key so that lines could be reduced hugely.

5. 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).

MySQL
1) Concurrency. We used connection pool to do this part. Connection pool contain multiple pre-connection which were connected before the client send requests.
2) Search for primary key. Primary key is unique in each table, mysql automatically build index on primary key in order to improve the effciency of searching.
3) Change query_cache_size. Query_cache_size cache the query and it's answer into memory which can be retrieve fast when request again.

HBase:
1) Concurrency is already implemented by HBase properties. The class Hconnection is concurrency after Hbase 0.94, and original Hconnectionpool is deprecated.
2) Search for rowkey. Rowkey is similar to primary in MySQL which is unique in each table and hbase automatically buid index on it in order to make search operation efficiently.
3) Change block size for Q2. The block size is the minimum size of block stored in memory. Since the answer of single request in Q2 is limited, we tune down the block size.
4) Split regions. Split table into different regions can maximum the usage of hdfs.

6. 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).

	Submission id	Without optimization	With optimization
MySQL	Connection pool	Id 70926 with 0.3 score	Id 71053 with 46.1 score
	Search primary key	Id 71173 with 0.32 score	Id 71194 with 32.67 score
	Increase query_cache_size	Id 71184 with 0.89 score	Id 71194 with 32.67 score
HBase	Connection pool	Id 72636 with 11.24 score	Id 72646 with 21.13 score
	Search rowkey	Id 72501 with 7.48 score	Id 72541 with 21.81 score
	Change block size	Id 72646 with 21.13	Id 72656 with 32.62

		score	score
	Split regions	Id 85785 with 37.57 score	Id 90793 with 85.28 score

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

We do not implement insert/update requests. We use replication to handle multiple request at a time. Since it is difficult to handle put consistency in different server, we do not implement it.

8. 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 use JDBC to connect to Mysql server and Cloudera HBase library to connect to HBase. JDBC is part of the Java Standard Edition platform, from Oracle Corporation. It is a industry standard for database-independent connectivity between the Java programming language. Cloudera HBase library is the standard library connect with Java and HBase on Cloudera hadoop. In Phase 1 we use Apache hadoop library on EMR, now we change to Cloudera since it give us more freedom to do configuration.

9. 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?

Yes. We split the table into different regions. In each region, we keep the balanced load request. A distributed file system can use every server as a file system in order to split the overhead.

10. 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).

Facebook: For storing username and password pairs, they should use an RDBMS. For storing images and videos in post, they should NoSQL .

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

MySQL: By using index on field which we need to search.
Hbase: By setting scan startrow and stoprow to avoid full table scan.

12. Did you use separate tables for Q2-Q3 on HBase and MySQL? How can you consolidate your tables to reduce memory usage?

Yes. We use separate tables for Q2-Q3 on both HBase and MySQL.

MySQL:
We put the two tables in one database. So they share the same configuration on database level. We also clear cache after Q2. But in live test, considering the mix query, we didn’t clear cache.

HBase:
We set the COMPRESSION of both table as SNAPPY and IN_MEMORY as true.

13. How did you profile the backend? If not, why not? Given a typical request-response for each query (q2-q3) 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 9x4 (Q2H to Q3M) table is one possible representation.

For MySQL, we used mysql shell and SET PROFILING = 1. Then we did some queries and SHOW PROFILES. We could see the time consumed in each query.
For HBase, we analyzed the CloudWatch monitering data and the charts on Cloudera Mangaer.
We could have the conclusion as follow.

MySQL Q2:

Factors	Latency percentage
Load Generator to Load Balancer	cannot measure
Load Balancer to Web Service	0
Parsing request	1
Web Service to DB	2
At DB (execution)	80
DB to Web Service	2

Parsing DB response	15
Web Service to LB	0
LB to LG	0
Total	100

MySQL Q3:

Factors	Latency percentage
Load Generator to Load Balancer	cannot measure
Load Balancer to Web Service	0
Parsing request	0
Web Service to DB	1
At DB (execution)	93
DB to Web Service	1
Parsing DB response	5
Web Service to LB	0
LB to LG	0
Total	100

HBase Q2:

Factors	Latency percentage
Load Balancer to Web Service	cannot measure
Parsing request	0
Web Service to DB	0
At DB (execution)	98
DB to Web Service	0
Parsing DB response	2

Web Service to LG	0
Total	100

HBase Q3:

Factors	Latency percentage
Load Balancer to Web Service	cannot measure
Parsing request	0
Web Service to DB	0
At DB (execution)	99
DB to Web Service	0
Parsing DB response	1
Web Service to LG	0
Total	100

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

We use 1 m4.large to develop MySQL back-end. The cost is about $1 or less.

And first we use EMR with 1 m4.large as master and 2 m4.large as core for HBase development. The total cost is at least $7. But the performance didn't worth so much money. So we utilize Cloudera to deploy HBase by ourselves. By using 3 m4.large instances are used for trial, the total cost is at most $1. After we could successfully deploy HBase with Cloudera Manager, we use 1 m3.medium and 6 m4.large to deploy HBase for live test. The cost is $0.787 per hour.

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

http://dev.mysql.com/doc/refman/5.6/en/range-optimization.html
http://stackoverflow.com/questions/5161291/mysql-optimizing-query-for-records-within-date-range
http://stackoverflow.com/questions/8881331/mysql-optimizing-the-search-on-range-of-records
http://dev.mysql.com/doc/refman/5.1/en/index-hints.html
http://www.tomcatexpert.com/blog/2010/04/01/configuring-jdbc-pool-high-concurrency
http://aws-labs.com/ubuntu-install-java-8/

https://www.linode.com/docs/databases/mysql/install-mysql-on-ubuntu-14-04
https://www.digitalocean.com/community/tutorials/how-to-use-mysql-query-profiling
https://www.percona.com/blog/2015/01/02/the-mysql-query-cache-how-it-works-and-workload-impacts-both-good-and-bad/
http://www.mkyong.com/java/find-out-your-java-heap-memory-size/
https://www.linode.com/docs/databases/mysql/tuning-your-mysql-database
http://serverfault.com/questions/78786/tuning-and-understanding-table-cache-in-mysql
http://webrewrite.com/how-to-check-ram-in-ubuntu-11-1012-04-using-terminal/
http://stackoverflow.com/questions/20259249/what-is-query-cache-prunes-per-day-in-mysql
http://www.hardwaresecrets.com/how-to-optimize-a-mysql-server/6/
https://easyengine.io/tutorials/mysql/mysqltuner/
http://stackoverflow.com/questions/3753504/mysqltuner-suggestions-and-changes-to-my-cnf
https://stanford.edu/dept/itss/docs/oracle/10g/appdev.101/b10826/sdo_objrelschema.htm#i1004730

[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
    b. The number and type of instances used and justification
    c. The spot cost for all instances used
    d. The execution time for the entire ETL process
    e. The overall cost of the ETL process
    f. The number of incomplete ETL runs before your final run
    g. Discuss difficulties encountered
    h. The size of the resulting database and reasoning
    i. The size of the backup

Q2:
We use EMR MapReduce model for ETL job. We use 1 master m3.xlarge instance and 9 core m3.xlarge for MapReduce. Since in that time our team can not spot 20 instances, we use on demand for 20 m3.xlarge plus emr for two hour is $13.54. For load time, we use m4.large spot $0.1 per hour to load sql and 6 m4.large spot $0.1 per hour to load data into HBase. The execution for entire Mysql ETL time is 1 hours and HBase about 1 hours. The overall cost is about $20 for ETL process. Since we change the database schema twice, we have 2 incomplete ETL runs before my final run. When we try to load data into mysql, we used 50GB disk and encountered the the problem no free space. Finally, we added a volume of 150GB and work fine. The size of resulting data base is around 90million row nearly 25GB, since we use userid_hashtag as primary key and the exacted sorted answer as column. We use 150GB volume as the backup.

Q3:
We use EMR MapReduce model for ET job. We use 1 master m3.xlarge instance and 9 core m3.xlarge for MapReduce. We used spot instances for 10 m3.xlarge plus emr for four hour is $3.78. For load time, we use m4.large spot $0.1 per hour to load sql and 6 m4.large spot $0.1 per hour to load data into HBase. The execution for entire Mysql load time is 3 hours and HBase about 1 hours. The overall cost is about $5 for ETL process. Since we change the database schema three times, we have 3 incomplete ETL runs before the final run. When we try to load data into mysql, we used 80GB disk and encountered the the problem no free space. Finally, we added a volume of 120GB and work fine. The size of resulting data base is around 40million row nearly 9GB, since we use userid_hashtag as primary key and the exacted sorted answer as column. We use 120GB volume as the backup.

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?

MapReduce. MapReduce support process and generate large data sets with a parallel, distributed algorithm on a cluster which contains multiple machines. We use 20 m3.xlarge machines for the extract and transform step, and 3 m4.large for load the data into Hbase. Since Mysql load does not support MapReduce, it take much longer time the load the data into database.

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

and why? How can you parallelize loading data into MySQL?

Yes, we used EMR to do streaming job. Streaming is faster, for mapper and reducer can work nearly at the same time. If using non-streaming, then all reducer need to wait till all mapper finished their work.

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 did not use any external tool such as pig to load the data. Although traditional tools provided by Mysql and HBase may take long time and effort to load data, we build several AMIs to avoid multiple loadings when we launch new instance.

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

MySQL. In HBase, we need to put the into HDFS to prepare for HBase loading and Prepare the HFiles for the HBase table. For MySQL, data could be loaded directly. When it comes to the HBase cluster configured by Cloudera manager, it took us a long time to figure out how to change the permission of Hfile to successfully load data.

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

Task 4: 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?

I think there should not be much different, for we used large-scale database systems to store those data, instead of saving them in plat files or RAM. So this design of front-end and back-end server is enough and it does not make much difference of performance.

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

Yes. We used a browser to send HTTP GET request, just like the load generator does, in order to see whether the result is bad or not. And observe the time consumption of handling single query. And we also develop our own load generator by write bash script and use multiple threads to generate HTTP request to our web application.

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

We would like to try alternative designs to avoid region server hotspotting. But we do not have enough time this time. Or use two small clusters and an ELB to handle HBase requests.

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

I think it is hard to solve the region server hotspotting problem in HBase. Even though HBase would self-adjust, but it's too slow and may have bad effects when query pattern changes frequently. We tried to split regions, salt rowkey with a prefix and simple distribution, but didn't get prominent improvement.

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

Q2:
We first considered save more than 10 million files on disk, so that we can just used the name of each file to query and it is very quick. But Linux file system does not allow us to create so many files. If the data is smaller, it might be a very quick and cool solution.

Q3:
Spatial index of MySQL maybe an alternative. But we didn't put it into practice since it's too complicated and we do not have enough time to explore it.

The split button for tables on Cloudera Manager console might be much more convenient than manually spliting. But it might also had bad impact on application performance since not all regions receive large amount of requests and if you do split on those regions with small amount of requests, the request distribution would only be more unevenly. And the split button on Cloudera Manager Console does split for all regions, regardless of the requests received.

[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**

]