**15619 Project Phase 3 Report**

**Performance Data and Configurations**

|  |  |
| --- | --- |
| Live Test Configuration and | Results |
| Instance type | m1.large |
| Number of instances | 7 |
| Cost per hour | 1.55 |
| Queries Per Second (QPS) | q1/q2/q3/q4/q5/q6/MIX[q1/q2/q3/q4/q5/q6/]  INSERT HERE:  score[90/11/26/0/17/3/MIX[135/3/6/14/7/2] ]  tput [12621.9/537.9/2657.4/3704.7/1334.0/220.2/MIX[4042.9/52.5/101.0/248.1/133.4/31.6] ]  latcy [7/92/18/13/37/227/MIX[12/476/247/100/187/794]]  corr [100/100/99/0/95/98/MIX[100/99/99/98/98/98]]  error [0/0/1/0/5/2/MIX[0/1/1/2/2/2]] |
| Relative Rank [1 - 91] : | Phase 1 :72  Phase 2 :74  Phase 3 :75 |
| Phase Score [out of 100] | ======  graded  ======  Phase 1:17  Phase 2 Live (selected, mention DB): MySQL, 33  Phase 3 Live: HBase,22  ======  others  ======  Phase 2 (Pre-Live):13  Phase 2 Live (dropped, mention DB): HBase 24  Phase 3 (Pre-Live): 34 |

**Team : Initializing**

**Members : Yang Wang, Ruixi Deng, Chao Li**

**[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 3. Make sure you spend a proportional amount of time. For instance, if you spent 30 hours building your system, you should spend 10 hours on the report. The best way is to do both simultaneously, make the report 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 use Java Servlet plus Tomacat7. We searched and compared different web frameworks in techempower.com and found java servlet is fast and easy to develop, so we chose this one.

Apache Tomcat 7 builds upon the improvements made in Tomcat 6 and implements the Servlet 3.0, JSP 2.2, EL 2.2 and Web Socket 1.1 specifications. In addition to that, it includes the following special properties:

* Web application memory leak detection and prevention
* Improved security for the Manager and Host Manager applications
* Generic CSRF protection
* Support for including external content directly in a web application
* Refactoring (connectors, lifecycle) and lots of internal code clean-up

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

In the phase 3, we put our front end server in the master node of Elastic Map-Reduce Cluster in order to reduce the total number of instances we used. In addition, this design will save the time of connection between the front server and the master node.

For the final submission, we used 3 clusters so we had 3 front end servers.

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

We had the following special optimizations for our front end system in order to make it adjust to the high throughput of the load generator:

* Deploy the whole front-end system in the master node of elastic map-reduce cluster in order to save time of connection and number of instances.
* Use Toncat 7 Dev version instead of official release version so that our developers have more freedom to configure the properties of Tomcat. For example, we can configure the jar file position when tomcat launched so that we can add many jars into the tomcat.
* Write a heartbeat request for tomcat in addition to all other required requests so that the load balance can get healthy information of the server through this special request

1. What did you change from Phase 1, 2 and why? If nothing, why not?

We made a lot of changes to the front end servers in order to meet the requirement of throughput since HBase is quite different from MySQL server.

1. We added cache system for the request sent from client. The strategy is double-edge sword. If the request are often same, which cause we need to respond with same context, the cache makes sense and accelerate the speed of throughput. But if the request are unique, cache miss will happen all the time. In addition, adding cache also cost extra spaces.
2. In the phase 1 and 2, we deployed one MySQL replica for every instance of front-end server. But in phase 3, the hbase worked as a cluster of master nodes and slave nodes. So we deployed the front-end server in the master node of cluster in order to save time of connection between front-end and hbase cluster.

1. Did you use an ELB for the front-end? Why, or why not? Condense your experience with ELB in the next few sentences.

Yes. We had 3 clusters of hbase, which means we had 3 set of front-end servers, so we used ELB to balance the load of every single cluster.

ELB is good because it prevents exhausting the resources of a single server. In addition, ELB is easy for us to manage multiple servers since ELB will check the health of every server and make the unhealthy servers out, which is beneficent for us in server cluster management.

However, this strategy also costs time because ELB needs to communicate with servers, which takes time. So we need to make the tradeoff when we decide if we are going to use ELB.

1. 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 to use one instance to handle Q1 and connect to several HBase clusters, but it does not work well since the throughput of Q1 is highest, and in such design, the front end instance’s load is heavy and cannot handle it.

In addition, we also tried to use one hbase cluster which had 6 slave nodes. And the master will take charge of Q1 and distribute the other queries, but it did not work well either because of the similar reason with the previous design.

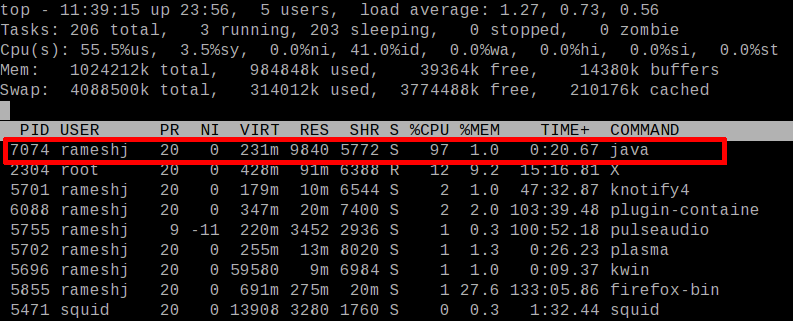
1. Did you automate your front-end? If yes, how? If no, why not?

Yes. We added the tomcat service into the linux init.d file so that the service will start automatically once the instance booted.

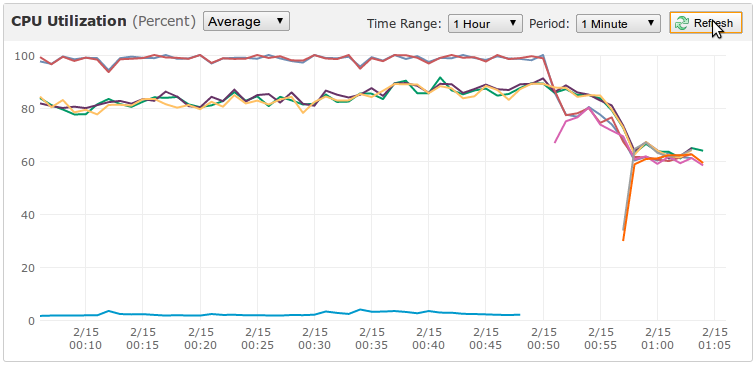
In addition, we wrote a script and added it into the cron table of Linux system in order to clear the tmp and logging files regularly .

1. 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 and Top command to monitor of the performance of our system.



Using Top Command to monitor usage of instance



Using cloud watch to monitor the usage of multiple instances

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

Since our front-end system is integrated with the EMR cluster, so we don’t have a separate cost for front-end system, it is all in the cost of EMR.

1. What are the best reference URLs (or books) that you found for your front-end?

The main resource we referred is web resource in StackOverflow and other sites.

# How to Install Apache Tomcat 7 or 8

# <http://www.excelsior-usa.com/articles/tomcat-amazon-ec2-basic.html>

# Cat in the Cloud: Apache Tomcat in Amazon EC2

# http://www.excelsior-usa.com/articles/tomcat-amazon-ec2-basic.html

1. Do you regret your front-end choice? If yes, what would you change in the way you approached Q1?

Basically we do not regret because Tomcat had many advantages except for sometimes uncontrolled logging system.

**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.

**HBase:**

For our schema design, we tried our best to achieve the following goals: clarity (easy to understand and use), simplicity, efficiency (good performance, especially read performance), and extensibility (have at least some ability for easier extension, though not required for this project).

For each Q2 - Q6, the schema of the table can be shown as below:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Query | Column Family | Row Key | Identifier | Cell Value |
| Q2 | q2 | userId + tweet time | tweetId, score, text | Id of the tweet, score of the tweet, censored text of the tweet. |
| Q3 | q3 | userId | retweetUserId | The list of userId’s that retweeted all the user Id’s. |
| Q4 | q4 | date + location + rank | hashtag, tweetIds | The hashtag corresponds to the rank and all the tweet Ids. |
| Q5 | q5 | userId | score1, score2, score3, total score | The score of the each three fields and the total score of the user |
| Q6 | q6 | userId | pictureNum, accumulatePictureNum | The number of the photos from THIS user, The second column represents the accumulative sum of pictures from ALL the users with smaller userIds. |

Some design ideas and reasoning for some of the schemas is shown below:

Because HBase do not support using two column as the row key, when it comes to situation where I need to user two fields to identify a record (such as Q2 and Q4), I simply used the concatenation of the two fields as the row key. We also tried to pre-process the information as much as we can, so that we can eliminate unnecessary and repeated lookups and calculation. This could help speed up the database. For example, in Q3 and Q6, we both record additional information in the database (the accumulative sum of all the previous user’s photos), this will decrease the number of the request to the database.

**For MySQL :**

for MySQL, our design is shown as followings:

|  |  |  |  |
| --- | --- | --- | --- |
| phase | fields | structures | how to specified one row |
| q2 | user\_id;date\_time;tweet\_id;text;core | int unsigned; varchar(20);text;text;int | we make indexes about the user\_id and date\_time fields. |
| q3 | origin;retweet | int unsigned;mediumtext | we make the origin field as the premier key, find the row by origin field. |
| q4 | date\_loc;tag;tweet\_id;rank | varchar(100);text;medium text;int unsigned | we make the indexes about date\_loc and rank fields. |
| q5 | user\_id;core\_1;score\_2;score\_3;total\_score | int unsigned;int unsigned;int unsigned;int unsigned;int unsigned; | we make the user\_id as the premier key, find the row directly by user\_id field |
| q6 | user\_id, photo;photo\_total | int unsigned;int unsigned;int unsigned; | we make the user\_id field as the premer key. And we use mysql command to find the rows in range, like the smallest user\_id larger than the specified user\_id and the largest user\_id that smaller than specified user\_id. |

For MySQL, I choose the design as the above because I think to the highest extend it satisfies the need of the queries, it is simple for the structure part of the schema and it’s efficient when searching for specified rows, like q6, we add the fields photo\_total because we have ranked the user\_id by ascending sequence and want to record the total photo numbers, in this way, if there is a range in a query, we can directly find the two rows of the upper bound m and lower bound n, then find the total number of m and n, so the value of total\_n minus total\_m will be the result we want, in this way, we avoid do the repeating work like calculating the total picture numbers, thus improving the throughput. And for some structures of the fields, we use the text or mediumtext instead of varchar(n), that’s because we don’t want to truncate the data when we do the loading work, if we choose simpler data structure and save time for loading, the correctness will drop down in the end. The correctness of the data is more important we think.

1. What was the most expensive operation / biggest problem with your 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.

**HBase:**

For query 2, the biggest problem was that there were too many records (i.e. two much rows) in the database. This makes the search very inefficient, and thus bring down the throughput of the database greatly. To resolve the problem, I make the following optimization, first of all, all the text stored in the database already have all the “sensitive” words filtered out. Then the front end do not need to do the filter the text, this would release the burden on the front end. Secondly, because the number of rows is very large, I pre-splitted the rowkeys into regions rather than let HBase do the region split for me. This would reduce the possibility for hot-spot, and ensure that the data are almost evenly distributed among the clusters. We did not have the database without these optimizations, hence we could not provide data for performance of this improvement.

For query 3, our original schema only records all the userIds that has retweeted the current user. Using this schema, in order to add the parenthesis correctly, we need to search through the whole database for every user in the retweet list. This can be very inefficient. To solve that, we did another round of data processing, the main purpose of which is to add all the needed parenthesis for the userId. With this change, we significantly decreased the number of search needed for each query. However, because of the size of the data, we used a m1.large machine for the parenthesis adding job and allotted a very large portion of the memory for the Java Virtual Machine.

For query 4, the problem is that for the same date and place, there might be a lot of different hashtags with different ranks. Storing all the hashtags in the same row would make a really large and massive row. When fetching the data, all hashtags are fetched back all the hashtags corresponds to the date and place even if they are not of the rank we needed. Transferring data through network can be very expensive, and we could try to avoid the unnecessary transmission of data. To solve for this problem, we changed the row key from date and location to the concatenation of date and location and the rank. This improved the performance by a very large scale (shown below):

|  |  |  |
| --- | --- | --- |
|  | Before | After |
| Submission ID: | 38926 | 40370 |
| through put | 20.2 | 955.5 |
| latency: | 2446 | 51 |

For query 6, the biggest question is that how to find the appropriate information we need, while still trying to eliminate the searches needed in the database. Because we do not know how many lines of data is actually stored between the two numbers provided by the query, we possibly need to scan through a lot of lines, and the scan can be possibly very slow. To solve for this problem, we stored the accumulated sum of the all the pictures of the users that has a smaller userId than the currently user. Using this technique, we only need to conduct two scan operation, each with a limit of 1. The improvement in performance can be shown below:

|  |  |  |
| --- | --- | --- |
|  | Before | After |
| Submission ID: | 38919 | 41771 |
| through put | 361.6 | 921.0 |
| latency: | 136 | 48 |

**For MySQL :**

For q2 and q4, the biggest problem for MySQL is the charset problem, we used to use uft8 instead of latin1 to load the data into database, but we found the result was not that right when compared with the golden result, our text result always had a shorter ‘?’ strings, then we thought: why our result was shorter, maybe we should not do some truncating work before return the data back to the front end. So we changed to default charset again and got a longer result. Because the incorrect text result was all represented by ‘?’, we did not know what happened with the wrong text. Then one day, when we searched the information about charset format of MySQL on the Internet, we found there was a sub charset called utf8mb4 which can tackle with the emoji charset, so we suddenly thought that the wrong result was due to the truncating or the extending of the emoji char, so before we loaded the data into the dataset, we wrote the following command to revise the charset:

alter database Q2 character set utf8mb4;

set names utf8mb4;

And also I revised the my.cnf file to make the I/O of the MySQL all in utf8mb4 charset format. The way I revised the file was as the followings:

[client]

default-character-set = utf8mb4

[mysql]

default-character-set = utf8mb4

[mysqld]

character-set-client-handshake = FALSE

character-set-server = utf8mb4

collation-server = utf8mb4\_unicode\_ci

init\_connect='SET NAMES utf8mb4'

And the following comparison tells us the correctness improved after we did the revisement:

|  |  |  |
| --- | --- | --- |
|  | Before | After |
| Submission ID: | 25535 | 25596 |
| through put | 231.0 | 244.5 |
| correctness | 66.00 | 100.00 |

1. Explain (briefly) **the theory** behind (at least) 10 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).

1) Using an external cache for frequently accessed data. This is mentioned by a lot of references, even in the official document provided by Hadoop. (Check this out:<http://hbase.apache.org/book/perf.reading.html> ). In our front end, we have a “cache” for each query that caches a certain amount of data. This helps improve the performance. However, when using the custom cache, it is very important that you limit your cache to a certain size and implement some delete policy when the cache is full (LRU, random delete, etc.). Otherwise, the cache will eventually drain your memory.

2) Using an internal cache. Besides the external cache, many database also provide a cache associated with the database of better read performance. This is both the case for SQL and HBase. However, in HBase, the cache refers to the lines fetched with a single request. If your query has a good locality, then using the cache would be very efficient. However, because in our case, the queries are considered to be random reads, hence using this cache will only fetch a bunch of unnecessary data. So, in our case, we limit this cache size to be 0 to reach a better performance.

3) Indexing. Indexing is a very common and important strategy in relational database like MySQL. In HBase, however, indexing on a column is not possible because rows in hbase is dynamic. Instead, HBase automatically rearranges the rowkeys lexicographically. So, to enhance better read performance, we could re-design the rowkey and split the table. (I think some people have done something called a “secondary indexing” on hbase, but we did not have the time to dig through it L)

4) Filtering rows correctly and avoid scan seek. This rule is also very important in both MySQL and HBase. In MySQL, a well formed query request with proper filtering might have a great improvement on the performance. Similary, hbase also have a similar filtering strategy. When using the scan operation (used to retrieve a range of rows) on the database, it is recorded by the hadoop’s official doc to add a filtering on the query instead of doing a “scan seek”, which simply conduct a general scan on the database. This is very important for q4 and 6, because we need to conduct efficient scan on the database.

5) Distribute the database well. HBase is a distributed database based on the hdfs. It is highly recommended by a lot of resources to design hbase and split the rowkey in such a way that it is almost evenly distributed. By proper distribution, we could reduce the possibility of “hot spot”, where all requires are jammed and becomes the bottleneck of the database. (If you have a hot spot a well splitted database, then you might have non-random reads and might need to consider caching as mentioned above.) MySQL also have its distributed version, however, according to our labs in module 3, this is more of an improvement of write rather than read.

6) Making replicas. Making replica is not only a matter of security, but also a matter of performance as well. In the live test for HBase, we have three identical clusters, each with its own server. The clusters are connected by the ELB. Our MySQL database also implements a very similar design, and this have a very good improvement on the total throughput.

7) Implement concurrent read. HBase has very good support for random read. Hence, configuring the database well so that it enables more concurrently read requests.

8) Appropriate type for data. Using the appropriate data to store in the database would decrease the size of the database and enhance performance. For instance, using long/int instead of string could save 3 times of the speed.

9) Hardware configuration. Because we are only using the machine for database use, we can configure some of the hardware properties to enhance the performance. Although we cannot use instances of xlarge type, we could resize the available memory for database and JVM.

1. 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).

Some of the optimization I used and their result can be shown as below:

1) Using external cache:

Using an external cache improved our performance for all the queries.

Using Q3 for example:

|  |  |  |
| --- | --- | --- |
|  | Before | After |
| Submission ID | 31964 | 31993 |
| throughput | 713.5 | 1328.4 |
| latency | 69 | 34 |

2) Configuring internal cache size.

We configured the internal cache for q6, so that when searching from a startRow, we only retrieve the data we need and not transferring redundant data through the network. Before the improvement, we have a very large cache size, and we set it to 1 because we only need one row of data. The improvement for q6 using this optimization can be shown below:

|  |  |  |
| --- | --- | --- |
|  | Before | After |
| Submission ID | 38907 | 40392 |
| throughput | 197.2 | 337.3 |
| latency | 251 | 146 |

3) Making replicas for database.

When we use three cluster connected by an ELB instead of using one single cluster with a large number of slaves, we also seen some improvement in all the queries. When we use 2 cluster each with 1 master and 1 slave instead of 1 cluster with 1 master and 4 slave, with smaller number of instances, we actually receive a better performance for all of the queries.

Using q3 for example:

|  |  |  |
| --- | --- | --- |
|  | Before | After |
| Submission ID | 31993 | 32036 |
| throughput | 1328.4 | 1503.1 |
| latency | 34 | 32 |

4) Split the rowkeys into regions.

Because the records for q2 is extremely large, I re-imported the data using a pre-defined split region. The improvement can be shown below:

|  |  |  |
| --- | --- | --- |
|  | Before | After |
| Submission ID | 31326 | 31337 |
| throughput | 94.3 | 158.4 |
| latency | 524 | 286 |

5) Appropriate type of data

For q5, our original implementation stored everything in the database as strings. We then figured out that the data are pure numbers and can be stored as strings. So we changed the data for q5, and the improvement can be shown below:

|  |  |  |
| --- | --- | --- |
|  | Before | After |
| Submission ID | 38919 | 42049 |
| throughput | 361.6 | 431.5 |
| latency | 136 | 104 |

6) For the result of revising the charset of utf8 to utf8mb4:

|  |  |  |
| --- | --- | --- |
|  | Before | After |
| Submission ID: | 25535 | 25596 |
| throughput | 231.0 | 244.5 |
| correctness | 66.00 | 100.00 |

7) For the result of revising the data gotten from the q3 about the way of adding brackets:

|  |  |  |
| --- | --- | --- |
|  | Before | After |
| Submission ID: | 23561 | 23652 |
| throughput | 760.8 | 332.1 |
| correctness | 41.00 | 100.00 |

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

I think the HBase will still work even if PUT requests are implemented. HBase have every powerful version control and support for concurrent read and write. So the database would be able to handle this query. However, the database might need to make some changes if put request is implemented. The regions might need to be re-designed to fit the needs for writes, since we might have incoming rowkeys that are not in our pre-visioned range, thus creating hot spots.

I think for the MySQL, the throughput will decrease a lot when it allows the PUT command since it will firstly find the revised location then update the whole database rows by rows, it just like adding a node in the center of the linked list. In this condition, I will make a MySQL cluster just like the small project told us. If there is some change in one database, the other database in the cluster will change at the same time. By applying such kind of replica, the throughput will maintain and it will be easier for us to revise the schema or the contents of the database.

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

For connecting to the HBase, we use the Apache hadoop HBase Java API. We chosed Java because HBase was developed in Jave, and it is the official API. It has the best support for the database. Also, using the Java API, we could use great HBase features such as bulk loading and MapReduce.

For MySQL, we use JDBC to connect with backend, it is easy to connect, handle exception and search for the data in the database then return to the front end. Because we use Tomcat as the front end and use Java as the language, we think JDBC is the best and the most compatible way to get data from the database, so we did not try other approaches and use JDBC at the first time.

1. How did you profile the backend? If not, why not? Given a typical request-response for each query (q1-q6) what percentage of the overall latency is due to:
   1. Load Generator to Load Balancer (if any, else merge with b.)
   2. Load Balancer to Web Service
   3. Parsing request
   4. Web Service to DB
   5. At DB (execution)
   6. DB to Web Service
   7. Parsing DB response
   8. Web Service to LB
   9. LB to LG

How did you measure this? A 9x6 table is one possible representation.

In our query, we made experiment and printed the timestamp and time elapsed for each period of the test, and calculated the following result.

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
|  | q1 | q2 | q3 | q4 | q5 | q6 |
| LG to LB | 10% | 7% | 11% | 9% | 6% | 3% |
| LB to web service | 20% | 8% | 3% | 2% | 4% | 4% |
| Parsing | 28% | 12% | 3% | 13% | 6% | 2% |
| Service to DB | 0% | 2% | 4% | 5% | 7% | 1% |
| At DB | 0% | 42% | 35% | 47% | 51% | 67% |
| DB to Service | 0% | 3% | 4% | 6% | 9% | 24% |
| Parsing Result | 12% | 11% | 2% | 7% | 7% | 2% |
| Service to LB | 20% | 8% | 3% | 2% | 4% | 4% |
| LB to LG | 10% | 7% | 11% | 9% | 6% | 3% |

1. 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).

For Amazon, a good reason to use HBase is for storing information about products. Because HBase is distributed, it is very efficient for E-commerce. Also, you might have many customers, so it is very important that the database have some support for concurrency. Thirdly, hbase has dynamic columns, which is very effective for adding new properties for certian products: you do not need to update the column for every other products as you must in MySQL.

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

For ET part, the cost is represented in the mapreduce work of EMR cluster, for phase 1, we costed 10 dollars, for phase 2, we costed 12 dollars since we remade q2 database data and for phase 3, we costed 8 dollars since the data we wanted was not that much. So the total cost for three phases was about 30 dollars including the verification cost.

For MySQL loading part, the cost was not that much because we needed only one instance to write the data into the database, q2 costed a lot since the size of the data was very large and it took lots of time to make indexes of fields, since we did not want our instance to shut down during the writing process, so we applied the on demand way. The total price for loading and schema revising for MySQL is about 10 ~ 15 dollars.

For HBase loading, the total cost is about 5 dollars because: 1, I used spot. 2, I used mapreduce, which is quick.

So the total cost is about 45 ~ 50 dollars for developing our back and system.

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

The resource for charset setting of MySQL:

<http://segmentfault.com/blog/ilikewhite/1190000000616820>

For deep understanding of charset format of MySQL:

<http://www.laruence.com/2008/01/05/12.html>

For HBase, some good resources includes:

Official documentation:

<http://hbase.apache.org/book/perf.writing.html#precreate.regions>

Stack Overflow (obviously)

IBM knowlege center (super good):

<http://www-01.ibm.com/support/knowledgecenter/SSPT3X_2.1.2/com.ibm.swg.im.infosphere.biginsights.analyze.doc/doc/bigsql_gentune.html>

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

**Task 3: ETL**

1. For each query, write about:
   1. The programming model used for the ETL job and justification
   2. The type of instances used and justification
   3. The number of instances used and justification
   4. The spot cost for all instances used
   5. The execution time for the entire ETL process
   6. The overall cost of the ETL process
   7. The number of incomplete ETL runs before your final run
   8. Discuss difficulties encountered
   9. The size of the resulting database and reasoning
   10. The time required to backup the database
   11. The size of the backup

**q2:**

a) The programming model for ETL job is similar to wordCount model, since we used mapper script to extract the information from twitter raw data and calculated the score and added stars in the reducer script, I used the sample given by the instructor to verified the small amount of data I have parsed from the EMR job, in this way, to the highest degree I can make sure the data I had processed was correct.

b) The type of instances I used for testing was m1 medium and I used m1 large instances to do the EMR job, since I used Python to write the mapper and reducer script, so medium instances may not be that powerful to do large scale job. That’s why I chose large instances.

c) I used 20 large m1 instances to do the EMR job and it was the maximum number I can use in EMR job.

d) The spot cost for the large m1 instances was about 1.6 cent and the spot cost for medium instances was about 0.8 cent. The total cost for the EMR job was about 8 dollars.

e) The entire ETL job was about 6 - 7 hours. That’s because I use Python to write the script, so it was a little lower when compared with Java script.

f) The overall cost of the ETL process was about 11 dollars, that includes the money we spent on the verification process and I redid the ETL job for q2 in phase 2 and phase 3 since there was something wrong when the data was loaded into the database.

g) The number was 2 - 3, the reason is that I did not know the standard utf8 charset did not support emoji, so I redid the EMR job by using the charset utf8mb4.

h) The difficulties include it always took a long time before I knew where I was wrong in EMR job. And it took me a lot of time to think why some chars for the text was truncated without given any hints. Because in the testing page provided by the course, it only showed the length of the text was not right and all the chars were shown as ‘?’. In the end, we found it’s the emoji thing and uft8 can not support emoji.

i) The size was about 30 G because the size of the database is proportional to the amount of data we stored in the database. And after the ET processes, we stored the text information into the database, so the size was a little large.

j) The time for backing up MySQL is represented in the making the image of the instances that stored the database, it was about 2 hours to make the snapshot of the instance. The backup time for HBase is about 5 mins.

k) For MySQL, the size of backup is represented by the size of the database, it was about 30 G. The HBase backup is about 62G.

**q3:**

a) The programming model for ETL job is similar to wordCount model, since we used mapper script to extract the information from twitter raw data and calculated the user\_id that retweeted the user’s tweet in the reducer script, I handle some small cases before I did the EMR job, so to the highest degree I can make sure the data I had processed was correct.

b) The type of instances I used for testing was m1 medium and I used m1 large instances to do the EMR job, since I used Python to write the mapper and reducer script, so medium instances may not be that powerful to do large scale job. That’s why I chose large instances. And for the brackets adding part, we wrote an additional Java file to do the job.

c) I used 20 large m1 instances to do the EMR job and it was the maximum number I can use in EMR job.

d) The spot cost for the large m1 instances was about 1.6 cent and the spot cost for medium instances was about 0.8 cent. The total cost for the EMR job was about 5 dollars. That’s because I faced few failures before we can make sure the data was right.

e) The entire ETL job was about 4 - 5 hours. That’s because I use Python to write the script, so it was a little lower when compared with Java script.

f) The overall cost of the ETL process was about 6 dollars, that includes the money we spent on the verification process and the money we opened the instance to do the bracket adding job.

g) The number is 1, the reason is that, the strategy I first used to do the ET job was incorrect because I should parse the data from the twitter itself.

h) The difficulties we faced includes that how to add the brackets after we got the data from the raw data. A large hashset and hashmap were needed.

i) The size was about 1 G because the size of the database is proportional to the amount of data we stored in the database. And after the ET processes, we stored just the user\_id data.

j) The time for backing up MySQL is represented in the making the image of the instances that stored the database, it was about 20 minutes to make the snapshot of the instance since the size of the data was not that large. The backup time is about 30 mins for hbase.

k) For MySQL, the size of backup is represented by the size of the database, it was about 1 G. The hbase backup size is about 2G.

**q4:**

a) The programming model for ETL job is similar to wordCount model, since we used mapper script to extract the information from twitter raw data and calculated the date, location, hashtags and tweet\_id list the hashtag is in in the reducer script, I handle some small cases before I did the EMR job and compared the result I got with the sample from the instructors, so to the highest degree I can make sure the data I had processed was correct.

b) The type of instances I used for testing was m1 medium and I used m1 large instances to do the EMR job, since I used Python to write the mapper and reducer script, so medium instances may not be that powerful to do large scale job. That’s why I chose large instances.

c) I used 20 large m1 instances to do the EMR job and it was the maximum number I can use in EMR job.

d) The spot cost for the large m1 instances was about 1.6 cent and the spot cost for medium instances was about 0.8 cent. The total cost for the EMR job was about 6 dollars. That’s because I faced few failures before we can make sure the data was right.

e) The entire ETL job was about 4 - 5 hours. That’s because I use Python to write the script, so it was a little lower when compared with Java script.

f) The overall cost of the ETL process was about 8 dollars, that includes the money we spent on the verification process.

g) The number is 2, the reason is that, the strategy I first used to do the EL job was incorrect because I should make the ranks of the hashtags after parsing the raw data in the reducer script.

h) The difficulties we faced includes that how to rank the hashtags according to the tweeter\_id list. Since there were not that many clear instructions in the writeup and there not exist useful explanation about the ranking before it’s too late.

i) The size was about 1 G because the size of the database is proportional to the amount of data we stored in the database. And after the ET processes, we stored just four to five columns of data.

j) The time for backing up MySQL is represented in the making the image of the instances that stored the database, it was about 30 minutes to make the snapshot of the instance since the size of the data was not that large. The backup time is about 35 mins.

k) For MySQL, the size of backup is represented by the size of the database, it was about 1 G. The hbase backup size is about 2G.

**q5:**

a) The programming model for ETL job is similar to wordCount model, since we used mapper script to extract the information from twitter raw data and calculated the scores for each item and the total score in the reducer script, moreover, I user the data from q3 since the third item of score is based on the dataset of q3. I handle some small cases before I did the EMR job and compared the result I got with the sample from the instructors, so to the highest degree I can make sure the data I had processed was correct. Because we used the data from q3 so multiple EMR steps were applied.

b) The type of instances I used for testing was m1 medium and I used m1 large instances to do the EMR job, since I used Python to write the mapper and reducer script, so medium instances may not be that powerful to do large scale job. That’s why I chose large instances. And the first step for parsing took much more time than step 2.

c) I used 20 large m1 instances to do the EMR job and it was the maximum number I can use in EMR job.

d) The spot cost for the large m1 instances was about 1.6 cent and the spot cost for medium instances was about 0.8 cent. The total cost for the EMR job was about 7 dollars. That’s because I used multiple steps to get the final result.

e) The entire ETL job was about 4 - 5 hours. That’s because I use Python to write the script, so it was a little lower when compared with Java script.

f) The overall cost of the ETL process was about 8 dollars, that includes the money we spent on the verification process. However, that did not include the money of reparsing the data for q3.

g) The number is 1, the reason was that I failed in the configuration part of the EMR cluster. I incorrectly set the output folder to be an existing folder.

h) For the technique level, I did not face any difficulties when doing ELB job, since it was very clear that how we can get the data from the raw data pool. In this question, I felt very excited because finally I can use the dataset from other questions. This combination is very useful for helping us understanding.

i) The size was about 1.1 G because the size of the database is proportional to the amount of data we stored in the database. And after the ET processes, we stored just four to five columns of data.

j) The time for backing up MySQL is represented in the making the image of the instances that stored the database, it was about 20 minutes to make the snapshot of the instance since the size of the data was not that large. The hbase backup time is about 40 min.

k) For MySQL, the size of backup is represented by the size of the database, it was about 1.1 G. the backup size for hbase is about 3 G.

**q6:**

a) The programming model for ETL job is similar to wordCount model, since we used mapper script to extract the information from twitter raw data and calculated the total photo numbers of specified user in the reducer script. I handled some small cases before I did the EMR job and compared the result I got with the sample from the instructors, so to the highest degree I can make sure the data I had processed was correct.

b) The type of instances I used for testing was m1 medium and I used m1 large instances to do the EMR job, since I used Python to write the mapper and reducer script, so medium instances may not be that powerful to do large scale job.

c) I used 20 large m1 instances to do the EMR job and it was the maximum number I can use in EMR job.

d) The spot cost for the large m1 instances was about 1.6 cent and the spot cost for medium instances was about 0.8 cent. The total cost for the EMR job was about 7 dollars. That’s because I used multiple steps to get the final result.

e) The entire ETL job was about 3 - 4 hours. That’s because I use Python to write the script, so it was a little lower when compared with Java script.

f) The overall cost of the ETL process was about 6 dollars, that’s because I redid the EMR job once for the user\_id without posting the photos.

g) The number is 1, the reason was that I redid the ETL job once. At first, I removed all the rows with the photo number equals to 0. It fitted the request of MySQL, but for Hbase, it was a little tricky to use scanner here, so I redid to leave all the data there.

h) For the technique level, I faced one problem that how to rank the user\_id after the mapper worked, the way I solved that was I refilled the user\_id to 10 digit number if it was less than 10 digit, in this way, the intermedium file after the mapper would rank the user\_id by int value instead of String value. After this, I can calculate the total photo numbers before this user in the database.

i) The size was about 1.2 G because the size of the database is proportional to the amount of data we stored in the database. And after the ET processes, we stored just four to five columns of data.

j) The time for backing up MySQL is represented in the making the image of the instances that stored the database, it was about 30 minutes to make the snapshot of the instance since the size of the data was not that large. The total backup time for hbase is 50 mins.

k) For MySQL, the size of backup is represented by the size of the database, it was about 1.3 G. The total size for the backup is 3G.

1. Critique your ETL techniques. Based on your experiences over the past 6 weeks, how would you do it differently if you had to do the same project again.

I think for the ET part of the raw twitter data, it was efficient enough because we spent as less time as possible and it was also flexible to revise the data we wanted since we usually stored a basic version of parsing the raw data, and we can do the EMR job again according to different needs of different questions. And we noticed that the first extraction was usually slow and the second parsing was much faster. So for the EL part, if I can do it again, I may test more small amount of cases to verify my method before applying it to real jobs.

For the L part, because I spend lots of time taking care of the charset part, so if I can do it again, I will firstly think about which charset will MySQL take and how to properly revise it.

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

First, the language we choose, I think Java is fastest language to do the ET job since Java is the favorite language of AWS. Then it’s the instance type we choose, although in this project, we are only allowed to use small to large type instances, we may save more money if we can use xlarge or 2xlarge instances, I have calculated before, the spotted price for large m1 instance is about 1.6 cents per hour, and it will take 7 hours (8) to do the EMR job, and for the same number of instances, the 2xlarge instances will only take 1.5 hours (2) to do the same job and it’s spotted price is about 6 cents per hour but with less EMR price, so in total, the 2xlarge instances will spend less money. Finally, the schema of Mysql is also important too, like the loading time for TEXT and MEDIUMTEXT are different when given the same rows of data.

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

Yes, I have used the EMR, and it was Streaming job, I think Streaming job is more faster than non-streaming job, because it allows you to do multi-steps mapreduce work. If you do the multiple EMR separately, the provision of instances will take time, the bootstrapping will take time, the checking configuration will take time, so there are lots of irrelevant time spent in the EMR job which will get the speed down.

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

Actually for MySQL, I did not use external tool to load the data, but I used local MySQL testbench to test small amount of cases with MySQL.

For HBase, we used the MapReduce Framework.

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

For MySQL, the time of loading is proportional to the size of the data, so if the dataset is small, it will be really fast to load the data. For the loading time value, I think MySQL is a little better than Hbase. Then, when we consider the data structure for loading, the MySQL is very restrict, you need to define the same delimiter sign for multiple fields. So, it’s may not that elastic when comparing with the Hbase. Moreover, we need to write a mapreduce script to store the data into Hbase, if the size of the data is too large just like q2, the time is about 2 hours, however, Hbase need to make backup in S3 bucket, so when comparing the total time for loading, MySQL use less time. And when comparing with the structure of loading data, if something wrong happened with the MySQL, it will directly truncate the data to structure with shorter bytes, but for Hbase, it gives higher tolerance.

However for HBase, even with large dataset, using the MapReduce framework, (as well as bulk load), it would still be able to have a very efficient write performance.

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

**General Questions**

1. What are the advantages and disadvantages of MySQL for each of the queries you’ve encountered so far? Which queries are better suited for MySQL (not HBase)?

For MySQL, the advantages are: the commands for searching specified queries are really a lot, and I can easily find one to fit my need. I remember in the question 6, I want to find a user\_id that was less than a number, I can easily use ‘group by desc’ to get it, the speed is fast, but for Hbase, we need to use Scanner which is much slower than the ‘get’ like method. But, the disadvantage for MySQL is that, we usually need to consider the type of charset when we load and extract the rows. Like the emoji, we need to use utf8mb4 instead of utf8 charset to do the loading work, it is not that elastic when comparing with Hbase about this point. For the queries, I think the queries that will take some range searching are more fitted for the MySQL, like the range searching is always a power function of MySQL.

1. What are the advantages and disadvantages of HBase for each of the queries you’ve encountered so far? Which queries are better suited for MySQL (not HBase)?

Because Hbase support concurrent read and write. It is suitable for situations where you need to handle concurrent read and write for the database.

It is also very good for database that need to pre-load huge data. Because it utilize the mapreduce framework, and the loading process is really fast.

It is also suitable for dealing with multi-language data. Because hbase store data as row bytes, there is much less nasty encoding issues as with MySQL.

1. For your backend design, what did you change from Phase 1 and Phase 2 and why? If nothing, why not?

For MySQL, I did not change the design for ELT part from the previous two phases, because for the data size level, the data we want is not that large and the time for loading into MySQL will take less time. Then, the strategy for searching is represented like making premier key and indexing as well, just like what the former two phases did.

I changed the schema for q2, so that there is less column family, and that all the information can be fetched with a single get request. This is more compact in space, and faster in terms of database I/O.

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 MySQL, if we need to tackle with more data size, I will separate one table to multiple tables, and each table takes care a range of search functions. In this way, the searching time for some queries will not be involved in a really large database, and if there is something we need to revise about the database, multiple data tables are more elastic.

For HBase, I would seperate the database through more clusters. Also, if there were some pattern in reading, we could re-design the rowkey, so that it enhances locality better.

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

We did not attempt to generate load on my own.

**More Questions (unscored)**

1. Describe an alternative design to your system that you wish you had time to try.
2. What were the five coolest things you learned in this project?
3. Which was/were the toughest roadblock(s) faced? What was the solution to that problem?
4. Design one interesting query for next semester’s students.
5. Did you do something unique (any cool optimization/trick/hack) that you would like to share with the class?
6. How will you describe this project (in one paragraph) on your LinkedIn / CV ?