# S15 15619 Project Phase 1 Report

**Performance Data and Configurations**

| Best Configuration and Results | |
|---|---|
| Number and type of instances | Q1: One m1.large<br>Q2H: -<br>Q2M: 12 DB Nodes (m1.small), 1 API Node (m1.large) and 1 management Node (m1.small) |
| Cost per hour | |
| Queries Per Second (QPS) | INSERT HERE: (Q1,Q2H,Q2M)<br>score[ 20, -, 0 ]<br>tput [ 15930.1, -, 1559.5 ]<br>latcy [ 5, -, 31 ]<br>corr [ 100, -, 0 ]<br>error [ 0, -, 0.01 ] |
| Rank on the scoreboard: | 35 |

**Team : (-_-)**
**Members : Abhishek Garai, Mithun Mathew, Namita Rane**

**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]

A.

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

A. First we utilized a single m3.medium, m1.medium to find the extent of throughput that could have been reached. Then we performed the tests with three m3.medium as m3.medium had better performance than m1.medium. But could not reach the desired throughput, expectedly because ELB was being used for load balancing. Further we tested with one m1.large to execute the test and thus could reach the desired throughput.

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

A. For the front-end instance we pre-warmed the cache by keeping the mappings of XYKey to ZKey values for the first 500000 serial numbers and later cached the request that were coming upto a limit of 1000000. Also, we implemented multithreading for performing the tasks concurrently specifically un-spiralling, intermediate key calculation for Q1 and sentiment score calculation, censoring for the map-reduce task.

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.

A. So, initially we tested with one instance of each m1.medium, m3.medium to test the extent of throughput that could be reached by each. Then accordingly we calculated the number of instances that would be necessary to reach the required throughput and put them behind the ELB. But, the ELB seemed to have slowed down the process and thus reduced the throughput significantly. So, we moved on to explore further available options for the load balancing. Load balancing is important in cloud because of the very nature of the structure i.e. there exists multiple physical infrastructure behind the hypervizer or virtualization software. So, its kind of mandatory that the the hypervisor should have sufficient capability to utilize the resources present behind it very efficiently by distributing the load equally for optimal performance. Also, load balancing in general provides better efficiency as every instance is being utilised to the optimum capacity independently.

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.

A. When ELB did not give the expected performance we started exploring other options available for load balancing and settled with nginx. We had performed our testing with other load balancers i.e. HA-Proxy and Varnish as well. From our test we concluded

that at high load by testing for longer time, nginx provide better throughput as compared to the other two. Although for lower loads it was varying and HA-Proxy and Varnish sometimes gave better results.

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

A. As far the automation of the front-end instance was considered, after launching the instance for the first time and installing the required softwares, we created an image of the instance and shared it among our team-mates. So, later we could just launched the instance and performed the task as the necessary softwares were already installed.

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

A. No we were planning to do it as we were not getting the performance. Later we did some i.e. instance utilizations for taking a decision on the type of instance to be used.

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

A. The total running costs for the front-end come to approximately $8.

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

http://undertow.io/

https://www.safaribooksonline.com/library/view/wildfly-performance-tuning/9781783980567/ch07s03.html

http://www.javacodegeeks.com/2014/01/entering-undertow-web-server.html

[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.

## Relational Database Design

The database design for relational database on MySQL required two iterations. Both designs have been described below.

### Design 1: tweet_table

| Column | Data Type | Description |
|---|---|---|
| user_id | DECIMAL(12,0) | user_id of the user publishing the tweet |
| tweet_time | VARCHAR(21) | time format in YYYY-MM-DD+HH:mm:ss |
| tweet_id | DECIMAL(14,0) | number format to enable sorting |
| score | DECIMAL(3,0) | score from sentiment analysis |
| censor_tweet | VARCHAR(200) | censored text (each tweet is max 140 characters long) |

**Pros:**

- Separation of fields would provide more flexibility during querying. For eg: SELECT * FROM tweet_table WHERE score = 5; is possible.

**Cons:**

- Requires concatenation to build the end response.
- For SELECT query, the WHERE condition has two clauses, WHERE user_id = '<user_id>' AND tweet_time = '<tweet_time>', composite index to be created

### Design 2: tweet_table

| Column | Data Type | Description |
|---|---|---|
| id | VARCHAR2(36) | U<user_id>T<tweet_time><br>concatenated version of user_id and tweet_time to make it more closely align with the request format.<br>front-end requires to input the separators 'U' and 'T' before querying the backend |
| response | VARCHAR(1000) | <tweet_id>:<score>:<censor_tweet><br>in the same format of the required response to |

| | | |
|---|---|---|
| | | reduce response latency from database side (to avoid concatenation during select query)<br>max of 6 or 7 tweets per given id<br>tweets sorted by tweet_id in the mapreduce phase |

**Pros:**

- Fields are closely aligned with the required request-response format.
- No further processing of response column is required after querying.
- Single index required on id column

**Cons:**

- Overhead in mapreduce phase to create this key-value pair after sorting through the multiple responses (value )for each id (key)

**References:**

http://stackoverflow.com/questions/179085/multiple-indexes-vs-multi-column-indexes

Design 2 was chosen for the following reasons:

- Ease of implementing it in map-reduce model
- Response format is same as response column - no overhead in processing for formatting response in SELECT query

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.

Querying was the most time consuming operation for query 2. As the DB got larger and the shards being managed multiple DB nodes, the query response was slower.
A single DB instance with higher capacity was launched - this ensured faster loading, but did not resolve the issue with slow querying time.

3. Explain (briefly) **the theory** 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).

- Using indexes: Using B-Tree index ensures faster search query from the database. http://www.scribd.com/doc/247092633/Indexing-in-Relational-Databases
- Disabling checks during load phase if not required (mapreduce output produces unique keys - disable unique checks in the database)
- Query optimization: reduce the number of joins and select only required columns as opposed to selecting all columns.

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).

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

For insert and update statements the design would work if the format is modified such that it is same as the columns in tweet_table.

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

The JDBC MySQL driver was used for connecting to the back-end database. It has an easy implementation which requires to specify the IP, port, database, user, password and table_name, which can be done in less than 5 lines of Java code. No other alternatives were considered.

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.

The latency for each level was not exactly measured. The DB execution took the most amount of time, while parsing DB response took least amount of time (since the response was already formatted)

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).

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

$15

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

MySQL documentation has very limited information and not even one example for different commands. StackOverFlow seemed to be more helpful for resolving MySQL errors and understanding usage of commands.

[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

Map-reduce programming model was used for extract and transform purpose. The model was used based on the key-value relationship which was derived from design 2's request id-response relationship from the database schema.

Loading was done manually using the MySQL LOAD DATA command, after retrieving the data from the S3 bucket.

    b. The number and type of instances used and justification

**EMR Job:**

m1.large: 1 master and 10 cores

2 (1 core and 1 master) m1.medium took around 10 minutes to complete the mapreduce job on one file.

The dataset contains 793 files. Approximate calculation:
800 * 10 min for m1.medium = 8000 min
Using m1.large, reducing time by a factor of 2 = 4000 min
Using 10 m1.large = 400 min ~ 6.67 hours (Note: this was the max time expected for the EMR job)

**Load Job:**

Sharding
12 DB Nodes (m1.small), 1 API Node (m1.large) and 1 management Node (m1.small)

Approximately 13 million rows were loaded from one output file. There were 17 output files ~ 221 million rows. To keep the row limit to less than 10 million in each shard, 12 DB nodes were initialized.

The API used was configured on m1.large to handle multiple incoming requests from the front end. 20 connection paths were defined from the API.

    c. The spot cost for all instances used

m1.small : $0.0071/hr
m1.medium : $0.0081/hr
m1.large : $0.0161/hr

d. The execution time for the entire ETL process

EMR Job: 5 hours 21 minutes (final run)
Load Job:

e. The overall cost of the ETL process

~ $10 for EMR job (includes testing and failures)

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

22 EMR runs (due to failures and unexpected results)

g. Discuss difficulties encountered

- Custom JARs dependent on third-party JAR for EMR job was not working as expected.
- Unavailability of the words AFINN and banned words on all cores (without distributed cache) gave unexpected results.
- Injecting special characters to denote mark the starting of a record, the start and end of column was rendered wrong because of the additional Combiner phase in the MapReduce
- Loading data failing due to DB nodes disconnecting without a reason that could be tracked.
- No way to track if the data was loaded completely into MySQL with LOAD DATA command - the cmd line just sits and waits for a long time with no ETA or percentage completion. Leaving the PC to run for long time, the SSH connection was closed. After SSHing back in there was no way to track if load was complete or failed. Counting the number of row in the table took over 15 minutes.

h. The size of the resulting database and reasoning
i. The size of the backup

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

- Loading into the local DB on the machine was faster than loading into a clustered system.
- The map reduce program output has only unique ids. Disabling the table's UNIQUE check enabled faster loading. Disabling binary logging also assisted in faster loading.

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

why?

Non-streaming EMR was used. The streaming EMR did not prove to be the best solution as the tweets had multiple lines and could not be read on a line by line basis. The wrapper class Text provided a good encapsulation for multi-line tweets and this was the main reason for using non-streaming EMR.

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

No

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

[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?

A. The front-end instance would be capable of handling a much higher load as we have implemented multiple strategies to optimize the front end i.e. we pre-warmed the cache and also had ongoing caching i.e. when the requests were received they were cached upto a certain limit and also implemented multithreading. Although not specifically known of the extent of further traffic it can handle.

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

A. We tested our code on the local machine for the EMR to verify whether it gives proper output. But as far the front end is concerned we did not generate any load from our end to test it.

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

A. In terms of designing of the undertow server, that we implemented, we were planning to implement the servlet version of the server because as per the documentation it would have a much better control on the different performance tuning parameters. Currently, we implemented the Builder API of undertow for Query1 and the manual server assembling for Query2. Also, we implemented the undertow through wildfly, we were planning to implement maven for building our project. Another, attempt that was planned was to analyse the xyKey that were present in the requests that were coming and store the exact mappings rather than for the first 500000 serial numbers.

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

A. For us the toughest roadblock was to reach the desired throughput for Query1. This was because we were misunderstanding the concept of the test durations available i.e. We executed the front-end code for 1 minute for all our initial tests and could not reach the desired throughput. But, when we executed the same code for 10 minutes with minor modifications like performing all the repetitive tasks like defining the date-format once rather than for each request, defining the Big-Integer XKey once etc., we could reach the desired throughput. The reason could be the cache may not have being getting warmed up during the short runs.

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

A. For the front-end instance in Query 1, since there were two operations being performed i.e. un-spiraling the encoded message and the calculation of the intermediate zKey from the xyKey present in the request, from the elements in the same request-url, we implemented multithreading to process these operations concurrently. This would expectedly highly optimize the performance during high load durations.

Similarly, for Query 2 ETL as well, since there were two operations that can be performed concurrently i.e. calculating the sentiment score and censoring the tweet-text,

we implemented multithreading to  optimize the performance of the program. Although we could not test them but expectedly it would provide optimal throughput during high load.