S15 15619 Project Phase 2 Report

Performance Data and Configurations

Best Configuration and	Results from the Live Test
Number and type of instances	HBase Live Test: - MySQL Live Test: 6 m1.large as the front end connected behind the load balancer having 1 m1.large back-end connected to each.
Cost per hour (assume on-demand prices)	HBase Live Test: - MySQL Live Test: $2.35
Queries Per Second (QPS)	INSERT HERE: (Q1,Q2H,Q2M,Q3H,Q3M,Q4H,Q4M) (see table below)
Rank on the scoreboard:	Q1: 10 Q2H: - Q2M: 19 Q3H: - Q3M: 20 Q4H: - Q4M: 16 HBase Live Test: - MySQL Live Test: 19

	Q1	Q2H	Q2M	Q3H	Q3M	Q4H	Q4M
score	132.1	-	22.9	-	74.89	-	68.86
tput	19814.6	-	1388.2	-	7641.8	-	4173.3
ltcy	4	-	35	-	6	-	11
corr	100	-	99	-	98.00	-	99.00
err	0	-	0	-	0.00	-	0.00

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

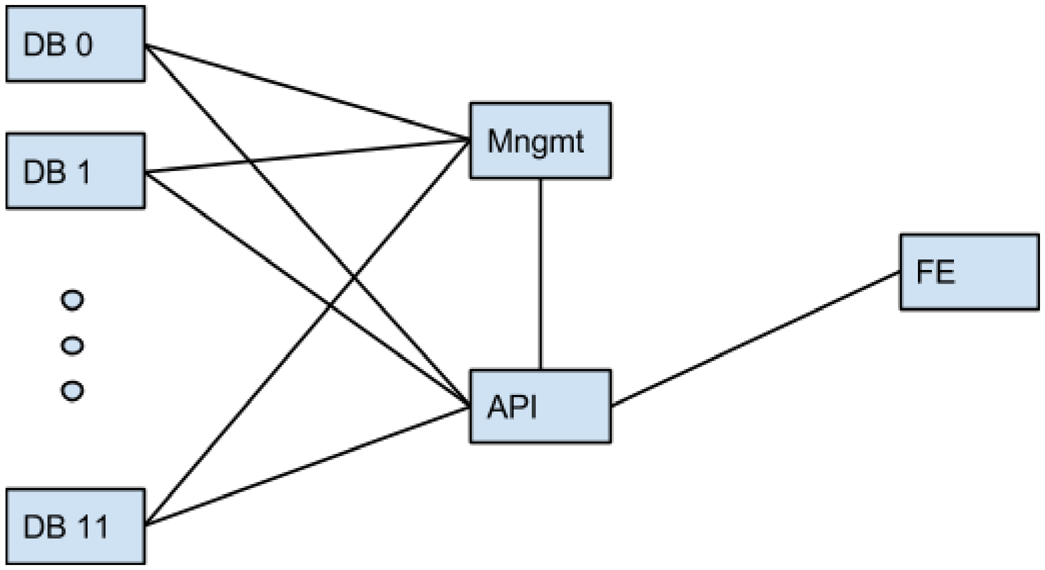
Rubric:

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

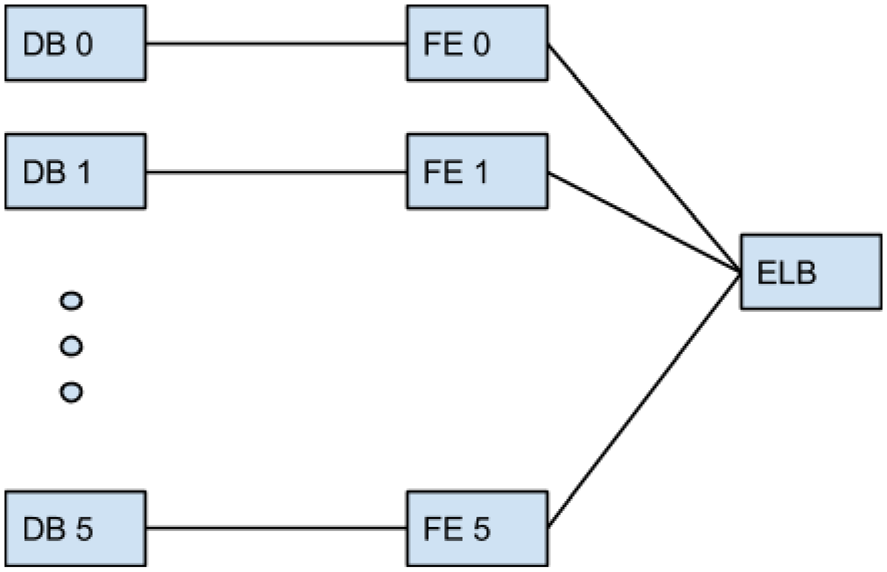
[Please provide an insightful, data-driven, colorful, chart/table-filled, <u>and</u> <u>interesting</u>

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)]

Phase 1 Structure (Sharded)



Phase 2 Structure (Replicated)



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].

   A. We used undertow web-framework as the front-end web framework for our project. Firstly, we chose undertow because of its best database access response per second for m1.large EC2 instances i.e. the front end instance type, according to the results from Tech Empower. Secondly, when we implemented other frameworks like java servlet for Apache Tomcat we attained lower throughput using the same number and type of instances i.e. undertow performed better for the optimizations that we could perform on both the types of frameworks. Properties are listed as below:

LightWeight- Various options for deploying
HTTP Support- Support HTTPS requests and responses
Web Socket Support-
Servelet 3.1-Web Servlet deployment
Embedded-Embedded using few lines of code also using wildfly
Flexible-Various configuration options

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

A. Since in phase 1 we could not reach the desired throughput for Query1 and also could not submit Query2 to test the latency and performance of undertow, we planned to change the webserver and performed testing with Apache Tomcat and java Servlet. But after test runs we found that undertow had a better performance as compared to servlets, resulting in higher throughput and lower latencies for equivalent configurations for both categories. Apparently we switched back to undertow for the further phase.

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

A. Since we had the restriction on the front-end instance type of m1.large, firstly, we performed the tests using the back-end instances of type m3.medium and m3.large.

Initially, we chose m3.medium because we thought that the back-end instance only had the database server running and only had to fetch the data and respond which would not require much processing power. Although, we did not take into account that the database fetch was a memory intensive operation. So, further we used m3.large hoping to utilize the memory available for fetching the data from the database. But due to the difference in the processing power of both m1.large front-end and m3.large back-end the performance of the operations went down, that we monitored using the cloud watch. And so finally we decided to settle for m1.large instance type as both the front-end and the back-end instance. Although we had a plan to use more number of back-end instances but restricted to six instances for front-end, back-end configuration with the ELB as we expected that it would cross the budget.

4. Explain any special configurations of your front end system.
   A. As a special configuration for the Front-end System we tried implementing the Asynchronous implementation of the JDBC but faced with a problem as in the requests were sent to the back-end server asynchronously and it was returning the responses, we checked it displaying the responses in the console, but the responses were not returned to the browser. Unable to fix the issue we continued with the normal blocking JDBC connectivity.

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.

A. So, after the experience we had in phase1 with the ELB we tried different configurations for distributing the load accross the back-end instances. So apparently, we tried the following configurations. 1) One Load balancer connected to multiple front-end instances and they in-turn connected to one back-end instance each. 2) Having a single front-end and multiple back-ends and distributing the load among the back-ends within the front-end program. 3) Deploying the front-end and the back-end within the same instance to reduce the network latency of connecting the back-end and the front-end. And, finally we settled with load balancing multiple front-ends each with a single back-end. Although we knew that this would not have been the best strategy, but due to lack of time to experiment we settled with it. 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, it’s 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.

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.

A. Our experience and experiments with different types of ELB was restricted to phase 1 and we used the ELB for phase 2. We explored the load balancers like HA Proxy, nginx and Varnish. Although ELB delayed the forwarding of the requests, but we could not configure the other load balancers to reach the expected performance after load balancing. For instance we tested nginx using two m1.large back-end instances and expected a higher throughput. But apparently we got a 16000 rps with one m1.large and 8000 rps with two m1.large instances behind nginx load balancer. We could not figure out the reason for this behaviour and hence settled with ELB as that was not reducing the throughput to such an extent.

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

A. In terms of automating the front-end instance, we experimented with wildfly and used the undertow server from within it, we could have automated the front-end instance to start the undertow server once the instance was launched. But later we imported the undertow library within the instance and launched the server manually each time. This was because although we installed wildfly for using undertow we could not figure out where to put the initialization script to start the server on boot-up.

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.

We were constantly monitoring the instances behind the ELB to ensure that they are healthy, up and running. The ELB parameters were also monitored during the live test. The no. of requests coming in for Q1 were high compared to that of Q2, Q3 and Q4. We also noticed the latency shown by the ELB was a good approximation of the latency on our scoreboard.

The number of HTTP 5xx were very low (some appeared towards the end of the test). Latency for Q2 and Q4 were considerably high (a little higher than we expected) compared to the individual submissions we made for each query after warmup. Q1 did way better than expected reaching 19k rps.

The CPU usage for the FE instances was around 20% during Q2 execution. During Q1 phase, CPU usage was much lower.

All our instances were working during the live test. We did not capture any screenshots or logs during the process (did not know that it is required for the report).

9. What was the cost to develop the front end system?
    A. For MySql, we launched 6 instances for front-end behind the load balancer. So the total cost approximated to $ 1.2.

10. 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 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.

MySQL DB Design

All three tables described below were our only design for phase 2. For phase 1, we had a key-value design for query2, which did not work out as planned, because of inefficient indexing: the key was unique and indexing did not help us decrease the latency. We decided to stick to a simplistic design for all three queries in phase 2.

Table: query2

Column	Data Type	Description
user_id	DECIMAL(15,0)	user_id of the user publishing the tweet
date_created	DATETIME	time format in YYYY-MM-DD+HH:mm:ss
tweet_id	DECIMAL(25,0)	number format to enable sorting
tweet_info	VARCHAR(250)	score from sentiment analysis and the censored text (each tweet is max 140 characters long)

Table: query3

Column	Data Type	Description
user_id	DECIMAL(15,0)	user_id of the user
response	VARCHAR(2000)	response containing the retweet relationship of the user - the response was created during the MapReduce job

Table: query4

Column	Data Type	Description
hashtag	VARCHAR(35)	hashtag to be searched for
tweet_id	DECIMAL(25,0)	number format to enable sorting
user_id	DECIMAL(15,0)	user_id of the user

date_created	DATETIME	time format in YYYY-MM-DD+HH:mm:ss

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

   Q2 - character encoding in the table to accommodate special characters and other languages
   Q4 - case sensitive collation for hashtag column. Initially it was case insensitive (by default), the index used for case-insensitive could not be used for case sensitive requests. Had to recreate the table with case-sensitive collation and create a new index on it

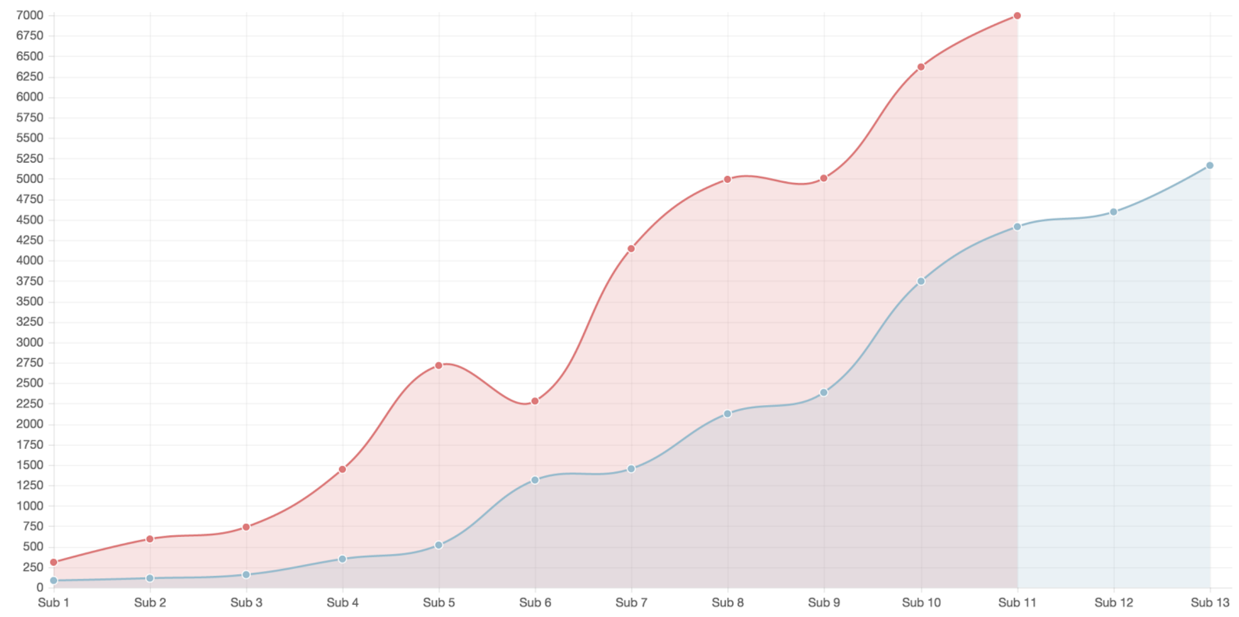
3. Explain (briefly) **<u>the theory</u>** behind (at least) 7 performance optimization techniques for databases. How are each of these implemented in MySQL? How are each of these implemented in HBase? Which optimizations only exist in one type of DB? How can you simulate that optimization in the other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

A.
- Implementing query cache
- Using EXPLAIN for optimization
- Avoid retrieving all columns in the table
- Limit the number of records
- Place the index on the search field (requested parameter)
- Index on join columns
- Always have a primary key
- Disabling unique checks during load phase if not required

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).

The graph below shows how the throughput increased steadily as we made multiple submissions one after the other for the same query. The exact reason for the increase in throughput when we made submissions one after another is not clear yet.

For the first few submissions, the test was run for 1 minute, then 3 minutes, 5 minutes and lastly 10 minutes. As we made these submissions which were spaced apart in time by minimal time, we noticed the following increasing trend:



Red: Q3
Blue: Q4
x - axis: Submission no. (increasing order of time)
y - axis: Throughput rps

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

A. The design would not work for insert and update requests. This is because the system works as 6 independent systems behind a load balancer (refer Phase 2 Structure). Even if the FE can handle PUT request, the load balancer sends the request only to one of the FEs, thus inserting/updating only one of the systems, which would lead to inconsistency.

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

A. MySQL JDBC driver was used to connect to the backend since that was used by us to first check the proper working of the code, logic and connection to the database. We tried Asynchronous JDBC driver with hopes of improving the throughput. However, we were not able to configure it successfully. In the meantime, we achieved a considerable rps with our old configuration.

7. Can you quantify the speed differential (in terms of rps or Mbps) between reading from disk versus reading from memory? Did you attempt to maximize your usage of

RAM to store your tables? How much (in % terms) of your memory could you use to respond to queries?

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

A. Could not implement Hbase.

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

A. Proper indexing MySQL databases based upon the type of requests made can improve read times.

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

A. We used separate tables for different queries for MySQL and Hbase. Consolidation of tables would have been possible to some extent for the queries based on the fact that the queries were dependent mostly on userid and specific columns from the tweet dataset

11. How did you profile the backend? If not, why not? Given a typical request-response for each query (q2-q4) 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 9x6 (Q2H to Q4M) table is one possible representation.

Profiling was done on the backend databases to analyze the improvement in read times with indexing on the tables created for different queries. After indexing the query response time was under 0.02s for all three queries.

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

A. We implemented 6 m1.large instance as our backend servers that approximated to $1.05 considering the on-demand price.

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

A. MySQl: http://www.tutorialspoint.com/jdbc/jdbc-sample-code.htm
https://code.google.com/p/async-mysql-connector/wiki/UsageExample
https://code.google.com/p/async-mysql-connector/
Hbase: http://hbase.apache.org/book.html
http://stackoverflow.com/questions/13906847/loading-csv-data-into-hbase

[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

MapReduce programming model was used for all three queries for the extract and transform phase. The reducer output was adjusted in such a way that we had a ",\t" separated file of fields, which could be directly used for loading using LOAD DATA command in MySQL

    b. The number and type of instances used and justification

EMR

Note: The no. and type of instances used for the EMR job were based on a basic approximation of time taken for the job on a dataset. They do not imply exact time calculations.

Q2

m1.large: 1 master and 12 cores

One file took 7 minutes with 1 m1.large master and 1 m1.medium core
793 files approximation = 793 * 7/60 = 92 hours
Dividing by a factor of 4/3 = 70 hours (adjusting between large+medium to large+large)
For 12 cores = 70/12 ~ 7 hours expected

Original time taken: 4 hours 27 minutes

Q3

m1.large: 1 master and 19 cores

One file took 2 minutes with 1 m1.medium master and 1 m1.medium core
793 files approximation = 793 * 2/60 = 27 hours
Dividing by a factor of 2 = 13.5 hours (adjusting between medium+medium to large+large)
For 19 cores = 13.5/19 ~ 42 min expected

Original time taken: 45 minutes

Q4

m1.xlarge: 1 master and 10 cores

One file took 2 minutes with 1 m1.medium master and 1 m1.medium core
793 files approximation = 793 * 2/60 = 27 hours
Dividing by a factor of 4 = 6 hours (adjusting between medium+medium to large+large)
For 10 cores = 6/10 ~ 36 min expected

Original time taken: 12 hours (This is the first time we used xlarge with hopes of getting faster results compared to large instances. But this turned out to be the least efficient EMR job we ever ran.)

Load
DB Nodes: 6x m1.large
FE Nodes: 6x m1.large

To handle multiple requests and achieve higher throughput, we configured six databases with its own front end. The load balancer sends requests to each front-end using round robin logic.

For phase 1, there was only one frontend which did not handle as many requests as expected and gave us low throughput. For phase 2, instead of using a MySQL replicated DB cluster, we configured the system to work as 6 independent services behind a load balancer, which worked out better than phase 1.

c. The spot cost for all instances used

m1.medium (test only)	: $0.0081/hr
m1.large	: $0.0161/hr
m1.xlarge (EMR only)	: $0.0321/hr

d. The execution time for the entire ETL process

Q2: 4 hours 27 minutes
Q3: 45 minutes
Q4: 12 hours

e. The overall cost of the ETL process

$24

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

Q2: 0
Q3: 6 unsuccessful attempts
Q4: 0

g. Discuss difficulties encountered

Q2: Putting the word lists on the distributed cache.

Q3: NumberFormatException when userid was beyond the limit of int

h. The size of the resulting database and reasoning

Total Size after Extraction and Transformation 67GB. This is because we are not storing all the contents that are present in the dataset rather we are extracting specific to the needs of the query.

i. The size of the backup

We did not create a backup for the DB. Instead we created an image of the instance on which the DB was installed and data was loaded.

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?

   A. Disabling the tables unique key checks and not defining a primary key for the MySQL table. Also disabling MySQL bin logging before loading are some of the approaches.

      We did not make any changes to the table structure or the data it held after the load. We had plans to change data within the table for which we got wrong results during submission. This could have been done by UPDATE statements to the database. But that would require updating the data individually within all 6 backend DBs, because they are independent of each other.

      Load times (approx.):
      Q2: 1 hour 35 minutes
      Q3: 15 minutes
      Q4: 25 minutes

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

   A. We used Non-streaming version of EMR.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 was one of the main advantage that we had in using non-streaming EMR.

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?

   A. For MySQL, we did read about Percona, but finally decided to use the LOAD DATA command because it was fast enough.

For phase1 Q2 we would have designed our database structure differently that would support both MySql and Hbase simultaneously.

5. Which database was easier to load (MySQL or HBase)? Why?
   A. For us MySQl database was easier to load as compared to hbase. The roadblock that we faced for loading hbase table was that we could not figure out the database structure that would be required to store the Hbase table in a key-value pair format for the Queries 2 and 4. Since, MySql did not have such an issue we could effectively provide as many columns required.

[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. We are bit skeptical about our web-service being able to handle a load that would be 10 times larger because we did not implement caching within our service that we feel could have been implemented for a higher throughput and load handling. Also, the tests failed for long 10 mins run and we could not figure out the issue for which we were facing the problem.

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

A. Yes. We wrote a JavaScript code to be executed from the browser that generated GET requests for a sequence of userids for Q3. This was done to warm up the ELB and test the response for the userids available in the Database. Although, most of the userids we sent as requests were not available in the data set as they were all sequential starting from 1 and specifically was used to warm-up the ELB to handle high-load and additionally check the response for the userids available.

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

A. We experimented with different combination for the overall structure of the web service as mentioned above. But apparently we could not realize the pros or cons for each of the design in terms of the performance that was offered. So, we were a bit skeptical later of which design to implement and finally implemented using the load balancer and multiple front-end, back-end pairs. But if we had time we would have tried to gain an insight on the performance of each and design the web service architecture further using the concepts of sharding, replications and auto-scaling as well. These were the things that we did not configure or had time to implement because of the other minor issues.

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

A. Understanding the achieved throughput for individual query submission. Submitting a query for 10 minutes always failed on the first attempt. Submitting the same multiple times for 1 minute (approx. 5 times), subsequently for 3 minutes(approx. 3 times) and 5 minutes (approx. 2 times) and 10 minutes (approx. 2 times) sequentially resulted in the throughput close to the requirement. This behaviour could not be interpreted. We speculated that since we were using ELB, it was the ELB that was warming up, but this does not fully satisfy the behaviour observed i.e. we need to always start from 1 minute run to run a test and achieve the desired throughput.

Also, optimizing the MySql query for the Queries 2,3,4 was another road-block, i.e. the optimization factors that should have been considered either within the program structure or the Database.

Another roadblock we faced was implementing hbase. Although we had a basic idea about data handling using hbase, we could not design the DB structure, load data and use the api to retrieve data from the hbase DB.

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

   A. For query 3, ETL was performed using a single run and the table in MySQL was stored two columns, userid (corresponds to the request) and response. This reduced the overhead on the MySQL backend and hence gave higher throughput for Query 3.

[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**

]