**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: 8 m1.large (1 master, 7 core)  MySQL Live Test:  1 Load Balancer  1 m1.large for the first 1.5 hours  Attached 6 more for the last 15. hours to ELB. |
| Cost per hour  (assume on-demand prices) | HBase Live Test: $1.40 (1 master and 7 core m1.large )  MySQL Live Test: $1.45 (7 m1.large and 1 ELB) |
| Queries Per Second (QPS) | INSERT HERE: (Q1,Q2H,Q2M,Q3H,Q3M,Q4H,Q4M)   |  |  |  |  |  |  |  |  | | --- | --- | --- | --- | --- | --- | --- | --- | |  | Q1 | Q2H | Q2M | Q3H | Q3M | Q4H | Q4M | | score | 50.93 | 0 | 7.97 | 3.19 | 0 | 4.57 | 9.16 | | tput | 7639 | 195 | 497 | 7327 | 12569 | 6851 | 1122 | | ltcy | 12 | 256 | 102 | 4 | 3 | 6 | 44 | | corr | 100 | 98 | 98 | 0 | 0 | 4 | 49 | | err | 0 | 0 | 0 | 82.33 | 0 | 0 | 0.11 | |
| Rank on the scoreboard: | (sorted by the column Q\* Best on the Score Board)  Q1: 41  Q2H: 43  Q2M: 40  Q3H: -  Q3M: 39  Q4H: -  Q4M: 44  HBase Live Test: 40  MySQL Live Test: 41 |

**Team : The RockStars**

**Members : Harini Rajendran, Manish Khemchandani, Shruti Anand**

**Rubric:**

**Each unanswered question = -7%**

**Each unsatisfactory answer = -3%**

**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 P1 Report)**

**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].**

(Taken from Phase 1)

Description of our journey from Report 1:

Query 1:

- The team kicked off the phase 1 query 1 by learning the expectations from phase 1 query 1 description. Since the team had earlier experience in Servlets, that is what we decided to go with. However, based on the information provided on<https://www.techempower.com/benchmarks/> and a meeting with our mentor who suggetsed that we use one of the top 5 frameworks for the purpose of scalability, we decided to use Gemini framework. When we actually tried to use it though, the poor documentation and online help caused us to switch to the next best option, Undertow. Briefly, we also considered Netty as a possible choice solely for the extensive documentation available, we rejected on the grounds of sub-par performance.

- Due to lack of documentation on undertow and after playing around with it for one day, time constraints the team caused us to switch back to Servlets for query 1.

Query 2 :

The move to Undertow

* Having a buffer of one week (colloquially called Spring Break by non-cloud students), the team focused on implementing query 1 using Undertow . Doing this helped the team in getting comfortable with the framework and found it quite simple to implement the existing servlets on Undertow which has embedded web server.

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

Phase 2 set back:

Having been so sure about using Undertow performance from the last two queries, we thought it would be easier for us to attain the same result for the phase 2. Its always said that under judging things lead to failures, and it struck us badly. From being on the top , we fell down to one of the bottom most positions. Just relying on framework and frontend caching didn’t help us to any extent. The queries were intense for phase 2, and we were unable to increase the rps by a greater margin, thus inhibiting us from scoring well. The team would still be sticking to the same framework, due to the experience that it has gained, but will be working on other compatible configurations which might raise the performance.

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

The instance type that we chose for our front end was 1 m1.large . Basically we would have got with more number of instances, but because of the on demand price constraint and the time constraint ( considering hbase was not working for us till the last moment, as visible from our score).

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

**HBase**: The team decided to use 8 m1. large instances under EMR cluster for the HBase front end. Out of these 8 instances, 1 was master and the rest 7 were slaves. The master instance was used to host the frontend while the rest 7 instances were used to host the backend (HBase).One of the key reasons for going with the creation of EMR clusters was the presence of pre built HBase. Also the data would be shared among the workers, and hence might help us in increasing performance, but since we were unable to finish on the hbase part , we couldn’t find ourselves in the “happy good performance” situation.

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

Yes this time the team was successful in creating a healthy ELB, after getting enough directions from piazza posts and understanding what we were missing upon. The team had last time failed to create a healthy ELB, as it was unable to send response to the ELBs. The ELBs were used for MySQL as it was very much apparent to us that using ELBs would be one of the trick to improve on the performance front.

The following are the major purposes that ELB caters to:

a. Distributing incoming traffic automatically or manually across amazon EC2 instances.

b. It also detects unhealthy instances and reroutes the traffic to the instances which are healthy.

c. Automatically scales it’s request handling capacity with varying traffic that it receives.

Though Amazon ELB helps in handling traffics, it doesn’t account for sudden traffics, and thus can only handle gradually increasing load patterns. Also there is a limitation in terms of time out time which is 60 seconds. This may be a problem when one wants to generate large files from EC2- backend. There is no start and stop option for ELB, it can only be deleted.

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.**

No since the team started off late due to internal issues and other course pressure, it didn’t find much time to experiment on this aspect. The team has decided though that one of the key takeaways would be to start early next phase, so as to retain back the old position, and also help in some experimentation part as well.

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

We didn’t automate on the front end front , still we did automate on few other parts like one of us who was responsible to write a script, created one so that HBase master node could distribute data throughout the tables of the slave nodes while also deleting temporary files to save disk space.

1. **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.**

Unlike last time, this time we did not profile the front-end. Due to a combination of bad planning and bad luck (our MySQL data got corrupted on the last day while we were taking a snapshot due to which we found ourselves loading the data at 3 pm on the day of submission), we were barely able to put the system together. We were not able to profile the front-end or even try out new optimizations after our 2-level cache failed to work as expected.

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

Our front- and back-end systems resided on the same machine for both HBase and MySQL.

* For HBase, 8 m1.large instances were created with the on-demand price of 0.175 per hour.

Total cost/hour = 1.4$

* For MySQL, 7 m3.large instances were attached to the ELB.

Total cost/hour = 1.225$

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

a. To find the best framework:

<https://www.techempower.com/benchmarks/#section=data-r9&hw=peak&test=db>

b. To implement undertow:<http://undertow.io/documentation/core/bootstrapping.html>

<http://docs.aws.amazon.com/AWSSdkDocsJava/latest/DeveloperGuide/java-dg-setup.html>

c. Configuring MySQL backend:

<http://www.percona.com/resources/mysql-ebooks/mysql-performance-tuning-volume-1>

[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.**

Answer: Query 2 schema for DB:

Column 1: userid# timestamp

Column 2: tweet id : sentiment score : tweet text.

Query 3 schema for DB:

Column 1: userid

Column 2: Relation as depicted in the writeup with each output line separated by ‘:’

\*,count1,userid1:\*,count2,userid2:+,count3,userid3:+,count4,userid4:-,count5,userid5:

1. **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.**

*Query 2*: Biggest problem: large amount of data . Problem to load specially in situations where the snapshot had become corrupted. It took more than 3 hours for just creating the index. There was no resolution that the team can come up with for this query.

*Query3*: Biggest problem : Complexity for calculation of different types of relationship between the users. Till the end the team was unable to come up with the initial logic for the reduce part specially.

How did we resolve it: At beginning we thought that we would extract the user ID and the retweeted userID fromt the data using ETL, and perform the entire relationship calculation in the front end. But then finally we added the logic of relationship calculation as per query3 description in the reducer.

*Query 4*: Biggest Problem: The database schema is as follows:

hashtag timestamp tweetID ,userID,timestamp

Since mySQL we didn’t set hashtag as the primary key hence it was easy for us to insert the duplicate hashtag values with different timestamp, tweet IDs etc. This schema did help us in querying the database from My SQL. But on the day of submission when we tried to use the same data and the schema for hbase the team members thought that the key ( first column- hashtag) for HBase should be unique, and hence we would need to again run the ETL to get a combination of data which would be combination of values making it unique. Realizing it 3 hours before the live test started made us stop from trying to run ETL again, and rather concentrate on improving the performance of queries which were working but had performance issues to be looked into. But during the live tests the team member trying to figure out the solution to this problem, so that we can make our stand better in next phase realized that the duplicate values get stored with their timestamps.

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

* **Caching** - Data is stored in memory which prevents an expensive I/O operation of reading data from the database. Caching can be done within the memory of the system or using a database cache.

This is implemented in MySQL by adding a couple of lines to edit the query cache size and the cache type in the configuration file of MySQL.

We implemented this in HBase by using the Scan class and its setCaching() method.

* **Indexing** - In indexing, data is sorted according to a key and a link to each row is maintained so that each access is orders of magnitude faster than a sequential search.

This is implemented in MySQL by using a CREATE INDEX query.

In HBase, the row key itself acts as the primary index. A secondary index can be created by maintain a lookup table which links to the row-key values.

* **Sharding** - Sharding is the distribution of data across multiple tables and nodes so that data accesses are significantly faster (because they are done in smaller tables).

In MySQL, this can be done by using a MySQL cluster rather than a stand-alone instance and setting the number of instances across which data is to be sharded in a configuration file.

HBase does auto-sharding that is it automatically distributes data across multiple instances when the table size becomes too large. Developers do not need to worry about setting up sharding.

* **Replication** - In replication, the entire dataset is present in multiple nodes so that the load is distributed among multiple nodes.

In MySQL, this is done by using a cluster and setting the number of instances across which data is to be replicated in a configuration file.

In HBase, this is done by making changes in the configuration file of the master node. However, it is important to note that replication is HBase is used to enhance availability rather than improve performance.

All the mentioned optimizations exist in both databases, although they are implemented differently as mentioned above.

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).**

As mentioned elsewhere, due to bad planning and misfortunes (our MySQL data got corrupted on the last day while we were taking a snapshot due to which we found ourselves loading the data at 3 pm on the day of submission), we were just trying to get the functionality together and were unable to apply optimizations on the code. One change that we did make was to create a datasource to better handle connections to MySQL, but we did not measure the rps value without that optimization.

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

(Partly taken from Phase 1)

The design would work for HBase but not MySQL for the reasons mentioned below:

For MySQL, we multiple instance each of which held the java front-end application as well as the database. This would allow a request to be routed to any front-end and the data fetched from the database on that instance. While this is fine for read requests, for write requests, only the database on one instance would get updated and thus, inconsistency of data between the different nodes would occur.

For HBase, we could simply implement a put method and the data would automatically be distributed across the clusters. We would need sufficient storage to handle the increased data size.

1. **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?**

(Taken from Phase 1)

We used a type 4 JDBC MySQL connector driver for connecting to MySQL since it provides the best performance of all classes of drivers. Moreover, team members were familiar with the use of that driver. For HBase, only one java driver was available which was in the form of JAR files provided by Hadoop. That was the one we used.

We explored other drivers for connecting to MySQL database, however based upon the performance offered by MySQL connector, we decided to go with it.

1. **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?**

MySQL:We used m1. large instances which had 8GB of ram and increased the primary partition size to 200 GB to load the mysql data for all the three queries. We weren’t able to figure out to increase the RAM size from 8GB to a larger size due to instance size restriction to m1. large. Also the team tried other techniques like attaching the volume, but it didn’t turn out to be a big help, in fact it slowed down the loading of data to the instances from s3 explorer.

1. **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?**

In our system, HBase did not use many features of HDFS. One use of a distributed file system is that commands to load data needed to be run only on the master node. The slaves were automatically populated with data by the master. Similarly, requests for data reach only the master. Data is fetched by the master from the slaves and sent back. In this manner, the cluster of slaves does not need to be managed separately.

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

The time required to perform scan reads can be reduced by prefetching the data and caching it. Certain assumptions can be made about the future requests based on the current request and that data can be pre-fetched and stored in the cache. Alternatively, only the current requests can be placed in the cache so that if they are requested again, an expensive I/O call to the database is saved.

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

Yes, we used separate tables for each Query for both, MySQL and HBase. Using separate table allowed us to optimize the data according to each Query. The tables for MySQL and HBase can be consolidated by modifying the schema and selecting specific columns from the table depending upon the type of query. Further processing would have to be done within the code to return the correct output. The drawback here is that performance would be impacted.

1. **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:**
   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 (Q2H to Q4M) table is one possible representation.**

Unlike last time, this time we did not profile the backend. Due to a combination of bad planning and bad luck (our MySQL data got corrupted on the last day while we were taking a snapshot due to which we found ourselves loading the data at 3 pm on the day of submission), we were barely able to put the system together. We were not able to profile the backend or even try out new optimizations after our 2-level cache failed to work as expected.

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

The cost to develop the back-end system would be the cost to create the data, load it and keep the instances running.

For ETL, the costs were:

* query 2: $1.034
* query 3: $1.034
* query 4: $1.470

Total: $3.538

For HBase, 8 m1.large instances were created with the on-demand price of 0.175 per hour.

Total cost/hour = 1.4$

For MySQL, 7 m1.large instances were attached to the ELB.

Total cost/hour = 1.225$

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

* <http://www.infoworld.com/article/2616674/database-administration/10-essential-performance-tips-for-mysql.html>
* <http://brettwooldridge.github.io/HikariCP/>
* <http://www.tocker.ca/2013/10/24/improving-the-performance-of-large-tables-in-mysql.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**

For query 1 we did not use any ETL job since it did not involve any twitter data analysis.

For the remaining three queries, Mapper-Reducer was used for ETL job.

**Query 2:**

Twitter data was picked up by Mapper and the operation for sentiments score calculation and text censoring was implemented in the mapper based on the criteria provided in the writeup. Reducer just printed them tab separated in a file for loading into the database.

**Query 3:**

Twitter data was processed by the mapper, it extracted the user id and retweeted user id and sent it to the reducer. The reducer calculated the relationships between the users as described in the write up and printed them tab separated in the output file for loading into the database.

**Query 4:**

The twitter data was processed by the mapper where the mapper extracted the hashtags, user id, tweet ids and timestamp from each tweet and sent it to the reducer. The reducer then formatted it as required for the database and printed them in a tab separated file for loading into the database.

The MapReduce framework was used by the team, as it would reduce the efforts and time to extract, reformat , apply operations on 1 tb of data. It was cost effective as after using 20 instances the time taken was just 1 hour and costed approximately 1 dollars.

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

For all the three queries we used m3.xlarge. Initially we were skeptical about using xlarge instances. Then after confirming on Piazza we decided to go with m3.xlarge.

For query 2 and query 3:

We used 24 m3.xlarge instances. Since these instances are very powerful, our ETL jobs got over in less than 1 hour.

For query 4:

We used 23 m1.large instances. It took around 1 hour for the ETL job to get over.

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

Each instance was 0.032$/hour for query 2 and 3.

Each instance was 0.016$ for query 4.

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

Query 2 and 3 took less than 1 hour each to get completed.

Query 4 took about 3.5 hours to get completed. By mistake we started query 4 processing using m1.large. We realized this after an hour, hence didn’t terminate the cluster. Pls don’t ask about the number of trials we took :)

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

Final run cost for

query 2 : $1.034

query 3: $1.034

query4: $1.470

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

Okay.. You have asked this question too :(

Query 2:

For query 2 things went well. We took 5 runs starting from 1 file and then 10 files and increased to the entire dataset in the 5th run.

Query 3 and 4:

This is when things started getting bad :(

Query 3:

11 runs before we figured out that there was no problem with the code so the problem was probably with the AMI version.

Query 4:

Ran ETL with 1 master and 1 worker node for 1 time with a single input file. Then 1 master and 4 m1.large nodes : 4 times to process 5 input files. And finally 1 run with 1 master and 23 workers to process the entire data.

* 1. **Discuss difficulties encountered**

Query 2:

a. The first difficulty that the team encountered was that the mapper failed around 6 to 7 times because the team missed on exception handling in mapper.

b. Sentiment score could be checked in local machine, but to check the logic correctness behind the mapper reducer, it was necessary to run mapper and reducer jobs on small files.

c. To come up with an output format which would be loaded to the schema of my sql, the team had to brainstorm and experiment to come up with schema which would reduce the querying time.

Query 3:

Query 4:

Processing a single file worked like a charm. But when we started running it for 5 files, map-reduce failed miserably. While the entire world was partying on friday night, we were struggling to get it work. Then we figured out that the error was in String handling part. We were creating new strings every time rather than reusing the same object, causing more memory constraints.

Once this was fixed, then the ETL job ran successfully on the entire data set.

As usual bad luck followed us this time too :p On the day of live test at around 2.00 PM, the only MySql instance which we had with all data loaded got terminated :( And the backup which was created was also corrupted. So we started loading the entire data in a fresh instance. Hence took time till 10 pm to load the entire data and index them. Hence we ran the first half of the live test with only one instance. Added 6 more instances to the Load balancer at the middle of live test after index are created in the database of the first instance. But nothing worked quite well.

* 1. **The size of the resulting database and reasoning**

Query 2: The resulting database was around 50 gb for query 2. Since we had to store the censored text it ate up a lot of space and money :( And every time took more than 2 hours for getting loaded into mysql. And more than 2 hours for creating index. This is one query which tested our patience every now and then.

Query 3: The resulting database was 3.9 gb.

Query 4: The resulting database was 6 gb.

Query 3 and 4 had very less data to store for each tweet. Hence did not occupy much space.

* 1. The size of the backup

The size of the backup was around 60 GB. Here we are just summing up the size of the database for query 2,3 and 4.

1. **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?**

We took powerful instances to run ETL faster. m3.xlarge instances came in handy and took only around $1 for the entire run. We optimized writing into mysql and hbase by creating a schema with no repetition of data across the fields. For HBase to load data into the clusters we wrote a shell script. The script loaded the tsv files in batch into the hbase cluster. Automating that task made hbase loading for q2 and q3 easier. We just created indexes on the MySQL tables after writing the data.

Query 2 : Took about 1.5 hours to get downloaded, 2 hours to load and 2 hours for indexing.

Query 3: 25 minutes to download from s3, 20 minutes to load and around the same time for indexing.

Query 4: Same as query 3.

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

As discussed in the above answers we used EMR to perform our ETL operations and also created HBase cluster. For ETL jobs we used streaming step. We used streaming because, streaming step involves processing the inputs at the same time using multiple mappers and reducers. We just used load data infile for loading data into mysql.

1. **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?**

The team used s3 explorer to download the sample input files from the s3 bucket for initial testing with 1 to 5 files. Except this we didn’t use any other external tools to load data into database.

If given a chance to do q2, phase 1 again, we would definitely strive hard to find a way to reduce the 5 hours of data loading and indexing time.

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

Both MySQL and HBase loading was easy this time since we had all the struggle of finding the HBase configuration in phase 1. The script which we wrote for HBase loading came in handy and made data loading easier.

**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?**

Answer : No to be very frank, our design failed us for this amount of data, thinking of restricting ourselves to just design , would so nothing but our laziness :P. Our team still can use the web services and the loading commands and the ETL code, but we would still need to work on the front end caching and the back end caching. Also we will need to work and reuse our knowledge for spatial and temporal caching in these scenarios.Doing this would help us coming up with a solution which would work for combination of data that would be pinging our serves.

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

Answer: The team though had decided to start early and work on having a load generator which would help us delve in these aspects as well, towards the end it became impossible for us . Due to increased complexity of the queries , our RPS had gone too low from what was anticipated, even for Query 2, so the team had to spend lot of time in researching the tuning parameters which might raise the rps for the current queries. But the team has decided to start early ( starting today itself ) for phase 3 and so if time and luck permits we would be working on experimenting with load generators as well.

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

Answer: First of all if we had time we would have tried to finish the HBase implementation for our query 3 and 4. Other than that if time permitted we would have tried to attach more ELBs to increase upon the performance . We weren’t able to do so till the last moment as our snapshot with the data had become corrupted on the morning of submission, and the loading was being done till 8 in the evening. Also one more thing that team strongly feels after going through blogs for enhancing performance specially in mySQL aspect was to use few techniques from Percona which emphasizes on fine tuning the my SQL backend in order to increase rps. We also found that implementing few new connection pools like Hikari Cp (brettwooldridge/HikariCP) would have helped us in raising our rps.

1. **Which was/were the toughest roadblocks faced in Phase 2?**

Answer:

**Misunderstanding on planning part :**The toughest part of phase 2 was to understand the complexity of phase 2 queries which were hitting our frontend. The caching was difficult for this phase. Also given that the team had only two weeks ( **Student syndrome**- out of which one week the team spent thinking that since phase 1 was done well, phase 2 would be easier) it was difficult for us to finish the whole mySQL and Hbase implementation timely. **Query 3 logic and configuration issues** :Also the implementation of query 3 logic was one thing in which the team was stuck . Not only the logic there were configuration issues while running EMR cluster ( the team members had chosen a wrong version) which was leading to job failures .

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

Answer: This time the team did try to attach ELB successfully , though only one due to data loading issues and for sure it raised the rps to certain extent. Also using prepared statement like last time compared to execute query also contributes to some extent to the difference in the performance. We tried using hikari config and backend caching, but since we were unable to integrate many nodes under ELB, we couldn’t notice much large rps increase, but trying this combination could be one more good idea.

**[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**

**]**