**S15 15619 Project Phase 3 Report**

**Performance Data and Configurations**

|  |  |
| --- | --- |
| Best Configuration and | Results from the Live Test |
| Choice of backend (pick one) | HBase/MySQL |
| Number and type of instances | Live Test: |
| Cost per hour  (assume on-demand prices) | Live Test: |
| Queries Per Second (QPS) | INSERT HERE: (Q1,Q2,Q3,Q4,Q5, Q6)   |  |  |  |  |  |  |  | | --- | --- | --- | --- | --- | --- | --- | |  | Q1 | Q2 | Q3 | Q4 | Q5 | Q6 | | score |  |  |  |  |  |  | | tput |  |  |  |  |  |  | | ltcy |  |  |  |  |  |  | | corr |  |  |  |  |  |  | | err |  |  |  |  |  |  | |
| Rank on the scoreboard: | Phase 1:  Phase 2:  Phase 3: |

**Team :**

**Members :**

**Rubric:**

**Each unanswered question = -10%**

**Each unsatisfactory answer = -5%**

**[Please provide an insightful, data-driven, colorful, chart/table-filled, and interesting final report. This is worth 20% of the grade for Phase 3. Use the report as a record of your progress, and then condense it before sharing it with us. Questions ending with “Why?” need evidence (not just logic)]**

**Task 1: Front end (you may/should copy answers from your earlier report-- each report should form a comprehensive account of your experiences building a cloud system. Please try to add more depth and cite references for your earlier answers)**

//Shruti

**Questions**

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 suggested 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.

- The team used the same framework for all the queries moving forward. The following the key challenges and issues that Undertow overcomes.

As with evolvement of Java EE 7 and the challenge to handle advanced features for example Web Socket API and HTTP upgrades like EJB over HTTP , Wild Fly made an important decision to release a new application server called Undertow. Undertow is flexible, fast web Server written in Java based on J2SE API.

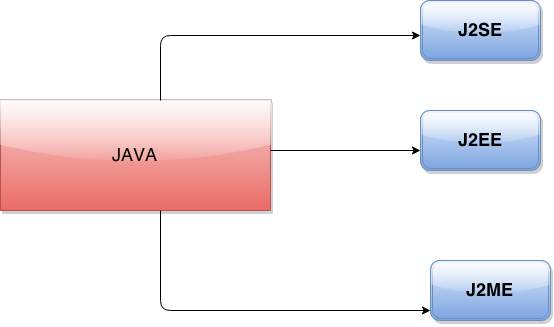


Figure 1:Java Platforms

As seen in the above figure there are three different platforms that Java support in order to address different requirements and challenges /frustrations faced by the developers. J2SE (Java Platform Standard Edition) is a platform which is used to address portability challenges by providing a platform for development and deployment of portable applications for desktop and server environments.

Why did Undertow come into picture??

Easy to use, high performance, efficient enough to meet the use cases that one can think of were some of the needs that led to Undertow release. Undertow is embeddable, and is easy to use. At the same time it is also well suited for application servers. It address performance issues and also possesses rich Java Capabilities. It can run new innovative APIs, but also standard APIs . It can run dynamic applications, but at the same time its light weightiness helps in replacement of native web server.

Architecture view point Undertow:

The flexibility in undertow architecture is encouraged by removal of concept of global containers and instead integrating undertow server by embedded application. An Undertow framework is basically composed of 3 things:

· XNIO worker instance( *The XNIO project provides a thin abstraction layer over Java NIO*)

· One or more connectors

· Handler chain to handle incoming requests.

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

Answer: As discussed in the above answer, the team still used the same framework for phase 3 as well. Since all the team members had become comfortable with the usage of Undertow, limited time budget and due to its highly flexible and performant behavior, moving to another framework didn’t seem to be an inviting or viable option to the team. The team had already seen the power of undertow framework while comparing the outcome of using servlets and running it on 6 instances and having a rps of 12000 after enough warming up exercises for the instances, while undertow with one instance resulted in rps of 12525.

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

Answer: The team chose m3. Large , considering that was the maximum instance limit a team could launch to adhere to given budget limit. Since the team had decided to have both the front end and backend on the same instance, it was necessary to have a robust and efficient processor. We could have launched an m1. Large instance, but considering m3. Large is later generation and cheaper than m1. Large the team chose the former. The team had started with by provisioning 3 instances and slowly in order to see the performance changes increased the number of instances. Once we crossed the limit of 6 and tried to provision 7 and 8 respectively, the performance decreased. Understanding that so many instances were becoming a bottleneck in performance, we reverted back and adhered to our decision of 6 instances.

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

Answer: The team used Hikari data source to increase the performance from the front end side while we retrieved data from the MySQL database. The team got to know about the usage of this configuration after going through the discussion on the following link:

### [The Blas from Pas: HikariCP (Connection Pool) with Pivotal...](http://theblasfrompas.blogspot.com/2013/12/hikaricp-connection-pool-with-pivotal.html) and

### [com.zaxxer.hikari.HikariConfig - GrepCode.com](http://grepcode.com/file/repo1.maven.org/maven2/com.zaxxer/HikariCP/1.3.5/com/zaxxer/hikari/HikariConfig.java).

Other than that the team tried to store all the data for query 6 in frontend, so that there was no requirement for hitting the database. But the team could only cache 5million rows using navigable set , considering it is a sorted data structure and allows data to be retrieved by giving a range (as required in q6) . But seeing there was not much change in performance if we use SQL databases to retrieve data, the team decided to stick to hitting database to retrieve data approach.

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.

Answer: Yes the team used ELB for the frontend as it thought it would help in increasing performance to a certain level if not drastically. While one of us experimented with one instance and the other one with ELBS having 3 to 6 instances, there was not much difference in rps that we observed. While with one instance the team could achieve rps of around 700 to 900 at a time, with ELBS after a few rps it would go upto 2000, even after attaching 6 instances and many rounds of warmups.The following are the major purposes that ELB caters to:

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

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

· 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.

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

Answers: The team thought of following ways to improve the performance:

· Storing data into memory (at least the small tables)

· Compressing the row data while storing in the tables.- The team tried to compress the tables by altering the already created tables with ROW\_FORMAT=COMPRESSED, but somehow the query took a lot of time to update changes. Thinking that we might corrupt the data if we try and make changes to table , we didn’t go ahead with this technique.

· Sharding- As taught in Project 3 the team tried to shard databases rather than replicating them over ELBs in order to improve the perfromance. But somehow it took us more than 8 hours to shard the data for query 2. Thus finding it too time consuming, the team left this approach as well.

· ElastiCache provided by the AWS- While researching on various many ways to improve performance , one of the techniques that the team came across was implementing Amazon ElastiCache. This is a webservice which offered by amaxon which helps in addressing performance issues by providing in memory caching (<http://aws.amazon.com/elasticache/>). But after learning how to use it ( and it comes free for t1. micro but since the datasets were too large , t1.micro couldn’t have helped us), due to cost and budget constraints the team didn’t use it.

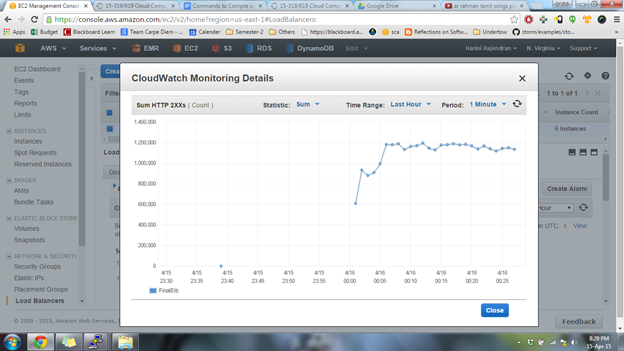
6. Did you automate your front-end instance? If yes, how? If no, why not?

Answer: We didn’t automate on the front end front , as team didn’t decide to use startup script, as the team figured out that we won’t be using more than 6 instances and since we could manually run frontend on all these servers the team didn’t emphasize on writing script to automate front end instance.

7. 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.

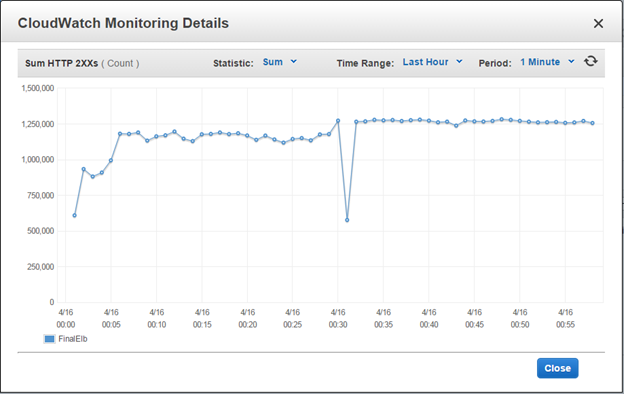
Answer: The team did monitor the front end by using default monitoring techniques provided by the Amazon web services.

The following figures represents the monitoring details for the various parts of the live tests:

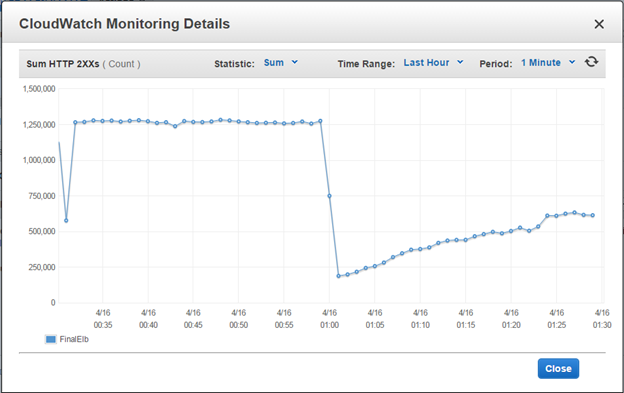


*Figure : Warmup CloudWatch monitoring details*

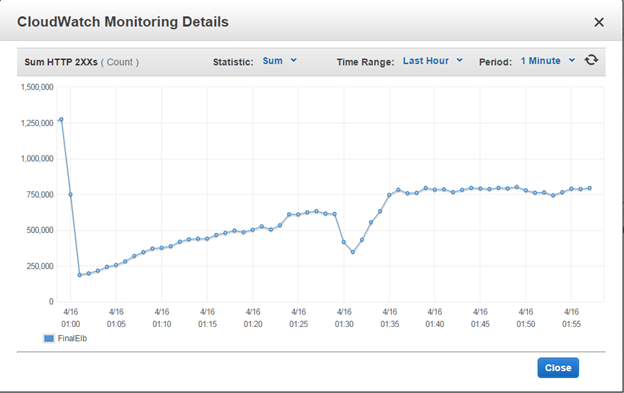
Query 1:



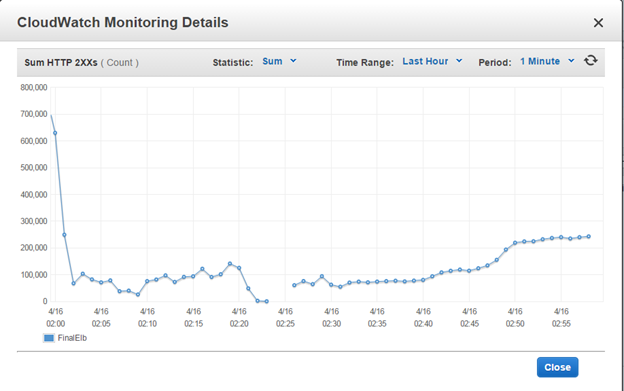
*Figure : Cloudwatch monitoring details for Query 1*



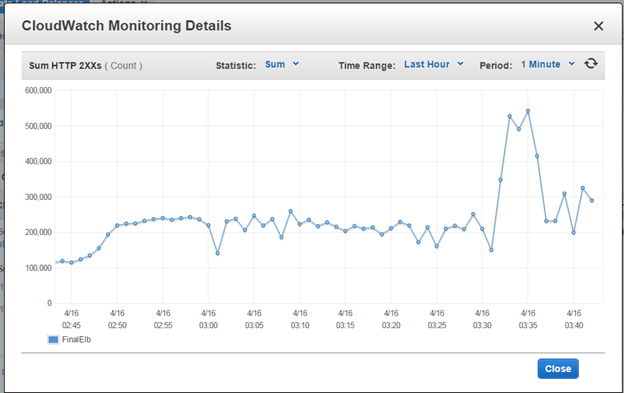
*Figure : Cloudwatch monitoring details for Query 2*



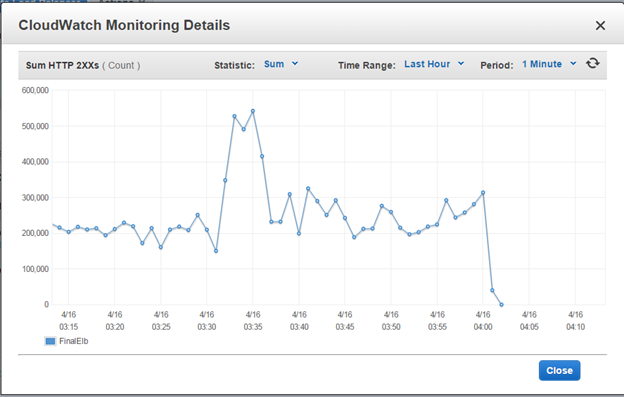
*Figure : Cloudwatch monitoring details for Query 3*



*Figure : Cloudwatch monitoring details for Query 4 and 5*



*Figure : Cloudwatch monitoring details for Query 6*



*Figure : Cloudwatch monitoring details for Mix*

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

**Answer**: Since we deployed our front end and databases (backend ) on every instance of ELB, in order to optimize and scale up the performance.

The team used 6 m3.large instances .

The spot price came upto $0.032\*6= $0.192

while the ondemand price was $0.145 \*6 =$0.87

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

Answer: Few of the urls that the team referred to for increasing performance are :

1. <http://sixrevisions.com/web-development/10-ways-to-improve-your-web-page-performance/>
2. <http://aws.amazon.com/elasticache/>
3. <https://www.techempower.com/benchmarks/#section=data-r9&hw=peak&test=db>
4. <http://undertow.io/documentation/core/bootstrapping.html>
5. <http://docs.aws.amazon.com/AWSSdkDocsJava/latest/DeveloperGuide/java-dg-setup.html>

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

**Task 2: Back end (database)** **Questions**

1. Which DB system did you choose in Phase 3? Why? Would any different queries for Q5 and Q6 have influenced you to choose the other DB?

We chose to go on with MySQL for phase 3. To be frank, we chose MySQL for phase 3 as we didn’t make much progress for Hbase in phase 2. We had little idea about Hbase and were more comfortable with using MySQL. In the limited time we had, we had to fix the data for q3 and have to get and load the data for q5 and q6. So we thought proceeding with Hbase would be risky. Hence, even if we had got different queries for q5 and q6, we would have mostly went with MySQL only. We would have made sure that the database schema which we make for those queries would be very efficient in such a way that it doesn’t require any front end processing.

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) **are required**.

Our MySQL Database schema:

Query 2:

userid#timestamp tweetid1:sentimentscore:censoredText\ntweetid2:sentimentscore:censoredText..

Query 3:

userid \*,count1,uid1\n....\n+,count4,uid4\n…\n-,count6,uid6\n…

Query 4:

hashtag timestamp tweetid,uid,timestamp

Query 5:

uid timestamp friendsCount followersCount

Query 6:

userid

Design Decisions:

We designed Query 2,3 and 6 schema in such a way that there is no processing required at the front end. Given the first column data from the request, we just retrieved the corresponding second column from the database and sent it back. This helped us and gave us a good performance during the live test. Whereas, for query 4 and query 5, since it involved the start and end timestamp, we couldn’t optimize the schema further. As a result we didn’t get the required rps for these queries.

For query 6, we decided we will load the entire data in memory instead of using database since data was very less after ETL. We implemented it but didn’t get enough time to test it properly. Hence we submitted the database version. We could have got good score for query 6 if we had used the main memory version. But we missed it due to time constraint again.

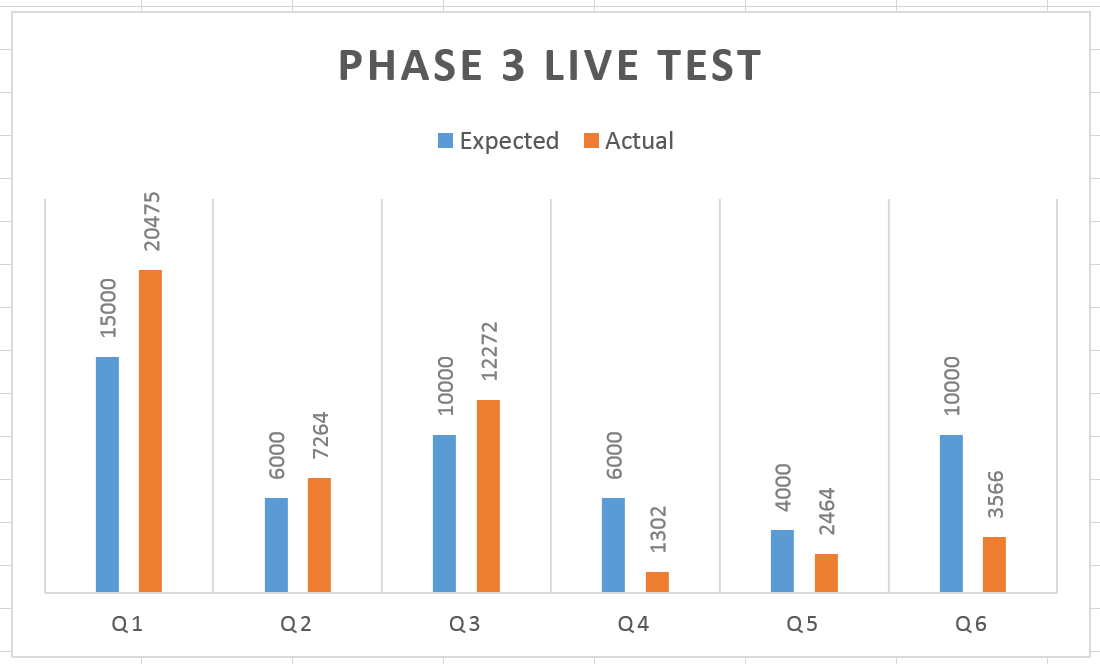
Design if we had gone with a different database:

We would have used the same schema only if we had gone for Hbase since we would have used the same tsv files which we got as a result of ETL for the first 4 queries.

We made sure that we are coming up with the best database schema we can think of in the first go. Hence after ETL we didn’t change the database schema for any of the queries. Hence we had only one iteration for the schemas. The following are the test results we got in the live test of phase 3.



Expected vs Actual:



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:

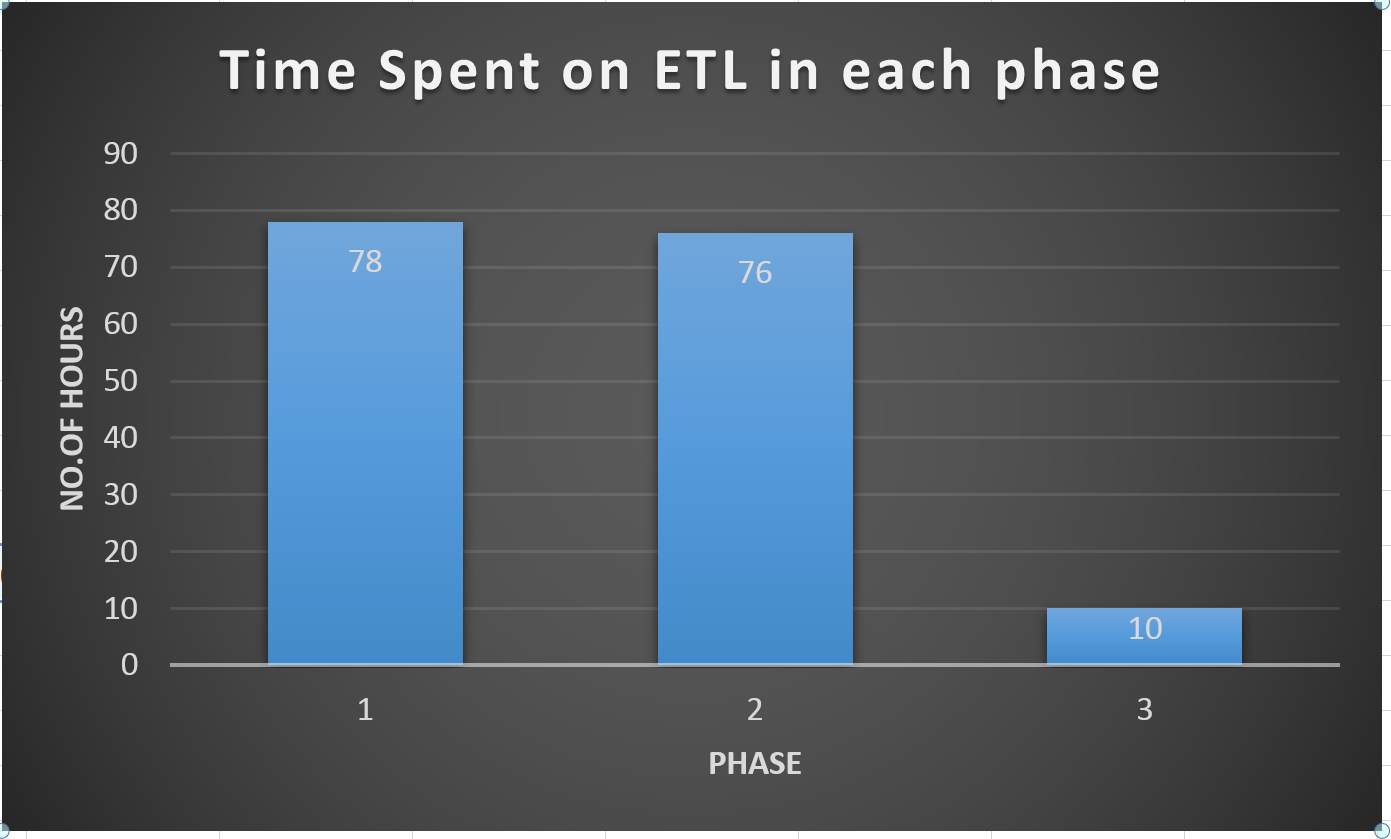
hashtag timestamp tweetID ,userID,timestamp

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.

Query 5 and Query 6: For query 5 and 6, since we focussed only on MySQL and since we got the logic right in the first attempt we didn’t face much issues. One problem with Query 5’s data was its size. It was around 14 GB and hence took more time to get loaded into SQL.

For query 6, the size of data was very less around 540 MB. In order to get a better performance, we had planned to store the data in main memory rather than accessing from the database. But we weren’t able to achieve this because of lack of time.

As the team got experienced using Map Reduce, ETL and undertow operations, the time we spent on getting the data significantly reduced as we moved towards the end even though the difficulty level of the queries were more or less same. We spent more time on optimizations and figuring out way to increase rps as we moved towards phase 3.



In phase 1, even though we had only one query for ETL we spent almost 80 hours to get the ETL done. It decreased by a considerable amount in phase 2 where we spent around the same time for ETL but we got the data loaded for 2 queries rather than 1 query. Whereas, the time we spent on ETL for phase 3 got reduced drastically.

1. Explain (briefly) **the theory** behind (at least) 11 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.

* **Innodb\_buffer\_pool\_size setting**- In MySQL, if the value of this innodb\_buffer\_pool\_size is less and if we still have more space left, then increasing the value of this variable will help in increasing the performance of MySQL as it reduces the number of Disk I/O required to retrieve data from the table.
* **Setting optimal value for MySQL\_Query\_Cache\_Size**- Starting from a small value of around 10 MB, we can tune this value to get MySQL perform better. But we should be careful and shouldn’t set it to 100s of MBs because then maintaining the cache will become an overhead and the performance will get reduced rather than getting increased.
* **Optimize Table MySQL** - In MySQL we can optimize the table by compacting the wasted space and creating the indexes again. This will reduce the database size and will increase the performance of the database.
* **Setting optimal hbase.client.scanner.caching** - In Hbase, this value in hbase-site.xml represents the number of rows to be scanned from the database if the requested data is not in memory. This will increase the performance, but might result in longer reads as huge block of data has to be read at one go.
* **Using machine with SSD** - Since database with only read requests have a great bottleneck on CPU I/O wait, SSD volume can be used to reduce this bottleneck.
* **Use profiling to learn the input pattern in case of regular queries** - If the request pattern is not random and we get the same kind of requests over time, we can examine these requests and understand the pattern and then implement efficient predictive caching to increase the performance of the database.
* **Optimal schema Design** - In both MySQL and Hbase designing an optimal database schema is an important pre-condition to achieve a good performance. Though we tune MySQL variables to give the best performance, a poor database schema will reduce the performance of the system as a whole.

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

**Answer:**

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? From Phase 2?

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. In all the three phases we used the same connector only.

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. Did you use separate tables for Q2-Q6? 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. What are the flaws you have seen in both DBs?
2. How did you profile the backend? If not, why not? Given a typical request-response for each query (Q2-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

We did not profile the backend. We spent more time in improving the correctness of the data for all the 6 queries and were trying to tune the settings of MySQL to improve the performance. We also got stuck in getting a good front end cache in place. So the team didn’t get time to profile the backend.

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: $3.102
* query 3: $3.102
* query 4: $4.410
* query 5: $1.472
* query 6: $1.472

We used 23 m3.xlarge for q2,q3,q5 and q6. For q4 we used 20 m1.large for ETL.

In addition to this, we used 8 m3.large instance attached to ELB for phase 2 and 6 m3.large instances attached to ELB for phase 3 with the mysql data loaded. So the cost of each instances was $0.032/hr.

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

[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 number and type of instances used and justification
   3. The spot cost for all instances used
   4. The execution time for the entire ETL process
   5. The overall cost of the ETL process
   6. The number of incomplete ETL runs before your final run
   7. Discuss difficulties encountered
   8. The size of the resulting database and reasoning
   9. The size of the backup

**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.

**Query 5:**

The twitter data was processed by the mapper where the mapper extracted the user id, timestamp, friends count and followers count for each tweet and sent it to the reducer. The reducer printed them in a tab separated file for loading into the database.

**Query 6:**

The twitter data was processed by the mapper where the mapper checked the location information for in each tweet. It sent out the user id and “0” separated by tab when location information is present in a tweet and sent out user id and “1” separated by tab when location information is not present. The reducer checked each row for the presence of location details. If all the tweets of a user doesn’t have location information, then that user id is outputted onto the file to be loaded into mysql.

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 more than 1 hour for the ETL job to get over.

For query 5 and query 6:

We used 23 m3.xlarge instances. Since these instances are very powerful, our ETL jobs got over in 40 minutes.

We wanted the ETL to be done is as much less time as possible. Hence we chose the number of instances in such a way that the job would be completed in less than an hour. Since the spot price of these instances were very less ($0.032/hr) selecting 23 instances didn’t cost much per run.

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

Each instance was 0.032$/hour for query 2,3,5 and 6.

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 :)

Query 5 and Query 6 also took less than 1 hour each to get completed.

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

Final run cost for

* query 2: $3.102
* query 3: $3.102
* query 4: $4.410
* query 5: $1.472
* query 6: $1.472
  1. **The number of incomplete ETL runs before your final run**

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:

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. We tried so many attempts with a single worker and single master instance and with 1 data file. The final run was for the entire data set.

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.

Query 5 and Query 6:

Query 5:

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 : 1 time to process 5 input files. And finally 1 run with 1 master and 23 workers to process the entire data.

Query 6:

Ran ETL with 1 master and 1 worker node multiple times since mapper was failing when it tried to check for location information. Then 1 master and 4 m1.large nodes : 1 time 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:

Getting the logic right this query was the most difficult part. Of all the queries we felt this query to be the trickiest one. And for some reason, the usual AMI version in EMR didn’t work at all when we ran the job. At the end we got so frustrated because there was no error in the logic and code, but still the mapper was failing. Then we tried with a different version in EMR and it didn’t fail after that.

Query 4:

Processing a single file worked like a charm. But when we started running it for 5 files, EMR 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.

Query 5 and 6:

For query 5 we didn’t encounter any difficulties. For query 6, we had to run the EMR for 3,4 times to get the logic for checking the location information right. For query 6, we first loaded the data into mysql with user id field’s data type as int. This resulted in overflow and all the user ids whose value is greater than the upper limit of int in mysql got truncated. Else we were getting 0 correctness for query 6. Then we figured out that this was the reason, created a table again with user id field as big int and loaded the data again. Since the size of data to be loaded is very less, it did not take much time for us.

In phase 2:

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.

In phase 3:

No such thing happened. All our data was safe till the end but the performance for q4,q5 and q6 were not as expected. For q4 and q5, we were more rps when we tested the application, but got reduced in the live test. On the other hand, q1, q2 and q3 performed extremely well in live run and rps was more than what we got during our testing.

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

Query 2: The resulting table 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 table was 3.9 gb.

Query 4: The resulting table was 6 gb.

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

Query 5: The resulting table was 14 gb.

Query 6: The resulting table was around 540 mb.

The entire database was 75 gb.

* 1. The size of the backup

The size of the backup was around 75 GB. Here we are just summing up the size of the tables for query 2,3,4,5 and 6. We didn’t note down the memory occupied by the indexes in each table :(

1. What are the most effective ways to speed up ETL? How did you optimize writing to your backend? 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.

Query 5: Took around 1 hour for the data to get downloaded, 1 hour to load and around the same time for indexing.

Query 6: Not even 10 minutes to download, load and index the data since the size of data was very less.

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

As discussed in the above answers we used EMR to perform our ETL operations and also created HBase cluster using EMR. 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?

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.

1. Which database has been easier to load (MySQL or HBase)? Why? Has your answer changed in the last four weeks?

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.

Comparatively we would say loading data into MySQL was prefered by the team because of the familiarity with MySQL than HBase. This is the reason for going on with MySQL in phase 3. Taking snapshot once the data is loaded and using the same snapshot to create multiple instances were all very easy in MySQL and till the end we didn’t figure out any such mechanism to store the Hbase data.

[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?**//Manish**
2. Did you attempt to generate load on your own? If yes, how? And why?
3. Describe an alternative design to your system that you wish you had time to try

**Answer**: The team did want to to try many solutions which it thought would have helped in improving the team performance , but couldn’t because of time , budget and complexity constraints. Sharding was one of the alternative which team wanted to implement and compare the rps with replication on ELBs. One of the team member wanted to use Amazon ElastiCache ( web service provided by Amazon with in memory caching )., but due to budget constraints couldn’t try it. Towards the end in order to scale up in terms of performance , we tried to alter table by compressing the rows , if there was more time the team wished to try compressing the rows of tables and see it’s effect on the performance.

1. Which was/were the toughest roadblock(s) faced in Phase 3?
2. Did you do something unique (any cool optimization/trick/hack) that you would like to share with the class?

**Bonus Questions**

1. Draw the Data Dependency DAG of fan-out requests (see P3.3 recitation [slide 10](http://www.cs.cmu.edu/~msakr/15619-s15/recitations/Recitation09.pdf) for an example of this graph) for the following sequence of events:
   1. Open the bonus page
   2. Search #beautiful between 2014-01-01 and 2014-12-31
   3. Click on the first 10 tweets
   4. Click on the first 10 users
2. Did you use any parallelization to speed up this data fetching at the front-end? Did you use any front-end web technique to make it appear faster for a visitor to the web page?
3. Show the network graph from your browser when 1a-1d are performed.