S15 15619 Project Phase 1 Report- *By The Rockstars*

Performance Data and Configurations

Best Configuration and Results	
Number and type of instances	Q1: 6 m1.large instances Q2H: 9 m1.large instances Q2M: 1 m1.large instances
Cost per hour	0.016 (spot price)
Queries Per Second (QPS)	INSERT HERE: (Q1,Q2H,Q2M) <u>Q1</u> score: 80 tput: 17354.8 latcy: 5 corr: 100.00 error: 0.00 <u>Q2H</u> score: 226 tput: 13683.3 latcy: 3 corr: 91.00 error: 0.00 <u>Q2M</u> score: 51.1 tput: 2810.5 latcy: 17 corr: 100.00 error: 0.00
Rank on the scoreboard:	Q1: 22 (sorted by the column Q1 Best on the ScoreBoard) Q2H: 1 (sorted by the column Q2 Best on the ScoreBoard) Q2M: 23 (Not available on the ScoreBoard but this is the approximate rank calculated based on the score of 51.1)

Team : The Rockstars

Members : Harini Rajendran, Manish Khemchandani, Shruti Anand

Rubric:
Each unanswered question = -5%
Each unsatisfactory answer = -2%

[Please provide an insightful, data-driven, colorful, chart/table-filled, humorous and interesting final report. This is worth a quarter of the grade for Phase 1. Use the report as a record of your progress, and then condense it before sharing it with us. Questions ending with "Why?" need evidence (not just logic)]

Task 1: Front end

Questions

1. *Which front end framework did you use? Explain why you used this solution. [Provide a small table of special properties that this framework/platform provides]*

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

1. 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.

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

Answer:

Query 1:

- Initially when the team decided to go with implementing servlets, team had to scale up to 6 m1.large instances, in order to achieve the required rps, which was greater than or equal to 15000 instances. m1.large was chosen because based on the results of an earlier individual project, m1.large provided excellent value (rps/cent), especially for spot instances. Moreover, since servlets provided less than satisfactory performance, we decided to scale up horizontally as well as vertically. As a result, the run time was 10 minutes, and the team was able to achieve 17354 rps, with no errors. Also the team had to do around 6 to 7 warmups to achieve the required rps.

Query 2 :
Web Service :

MySQL:Front end was installed on the same node as MySql. Though the team had decided to previously go with mySQL clusters, loading data on the cluster didn't workout. So the team decided to go with one instance. We thought that the combination of implementing mySQL query caching, indexing and front end caching would be enough to get the required RPS.

HBase: The team decided to use 9 m1. large instances under EMR cluster for the HBase front end. Out of these 9 instances 1 was master and the rest 8 were slaves. The master instance was used to host the frontend while the rest 8 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.

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

Answer 1:

Query 1:

Software configurations: Apache Tomcat

Logical Configuration: *

Query 2:

Used undertow which has embedded servers, so we had to only install java, mySQL and HBase , as per requirements stated in the query 2 description. Also to view large data files especially the ones which were output from the reducers and couldn't open in the Notepad++, we had to install Cygwin. The team did sharding using 5 instances while doing the loading part in mySQL .

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.*

Query 1: Yes, ELB was used for implementing the front-end. For the reasons mentioned above, the team decided to go with servlets for implementing the front-end. However, to meet the desired rps value, it became clear that we would need to scale up horizontally and vertically. Thus, we used an ELB to distribute the load. An ELB is used to distribute traffic amongst multiple nodes to ensure large amounts of traffic. It also allow one to scale up or down based upon the amount of incoming traffic. By employing the use of ELB, we found that even though we used servlets, we could handle more traffic by scaling up and thus were able to achieve the desired rps.

Query 2: The team didn't implement the ELB. Since EMR clusters can come with HBase preinstalled, we decided to go with those for the HBase part rather than ELB. For the MySQL part, we tried to switch from a single instance to a cluster after we realized that the combination of indexing and caching was not enough to handle the large load. But because of the number of issues we faced, we decided to stick with a single instance.

General Usage of ELB :

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.

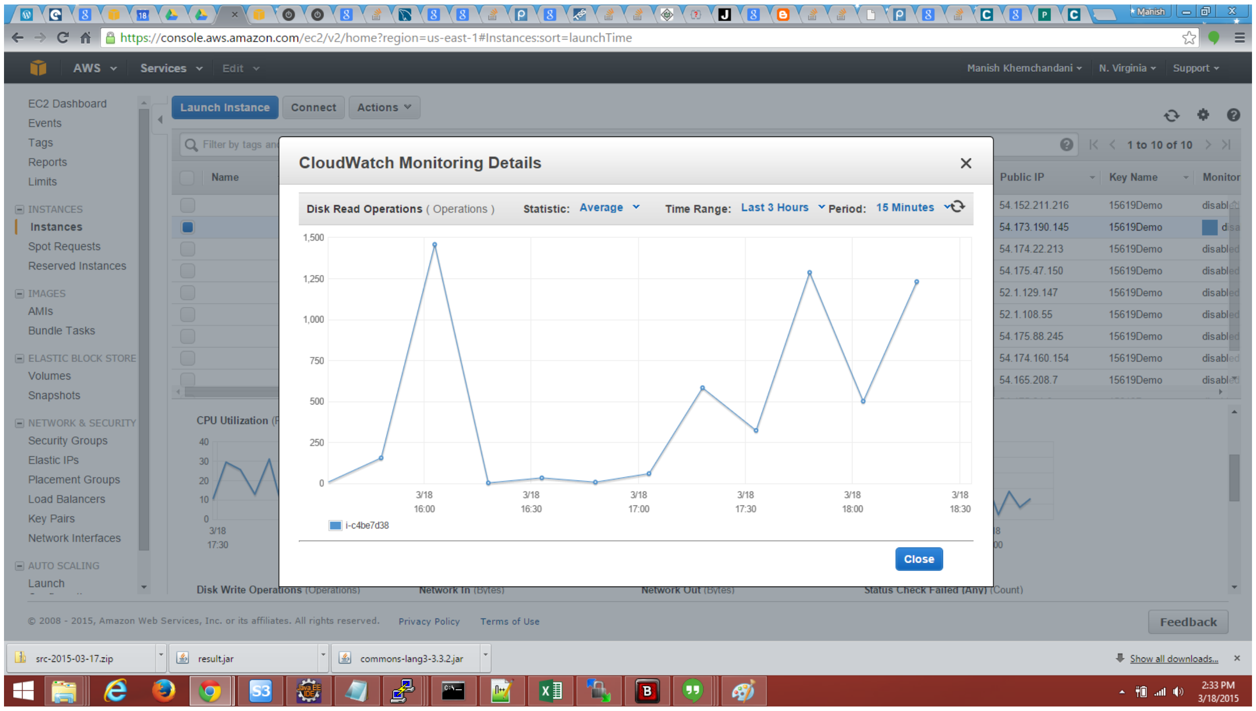
Though Amazon ELB helps in handling traffics, it doesn't account for sudden traffics, and thus can only handle gradually increasing load patterns. Also there is a limitation in terms of time out time which is 60 seconds. This may be a problem when one wants to generate large files from EC2- backend.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

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.*

   Answer: We used EMR as an alternative to ELB in the sense that a single master used to access data from HBase tables stored in 8 slaves. However, if we consider ELB as a way to distribute load over multiple nodes, then our EMR implementation may not be considered as an alternative to ELB.

6. *Did you automate your front-end instance? If yes, how? If no, why not?*
   Answer: Query 1: Added all startup scripts of tomcat to host the web service in the init script.
   Query 2: We wrote a script so that HBase master node could distribute data throughout the tables of the slave nodes while also deleting temporary files to save disk space.

7. *Did you use any form of monitoring on your front-end? Why or why not? If you did, show us the results.*
   Answer: In the front-end system for query 1, we did not use monitoring. For query 2, we monitored the disk read operations to check our performance over time.



Disk Read Operations (Operations) Statistic: Average Time Range: Last 3 Hours Period: 15 Minutes

Public IP	Key Name	Monitor
54.152.211.216	15619Demo	disabl
54.173.190.145	15619Demo	disa
54.174.22.213	15619Demo	disabled
54.175.47.150	15619Demo	disabled
52.1.129.147	15619Demo	disabled
52.1.108.55	15619Demo	disabled
54.175.88.245	15619Demo	disabled
54.174.160.154	15619Demo	disabled
54.165.208.7	15619Demo	disabl

© 2008 - 2015, Amazon Web Services, Inc. or its affiliates. All rights reserved. Privacy Policy Terms of Use

Graph1:CloudWatching Monitoring Details

8. *What was the cost to develop the front end system?*
   The front end (query 1) was implemented within 6 dollars. The cost for Query 2 consists of both front- and back-end costs.

9. *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
   c. 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. *Describe your schema. Explain your schema design decisions. Would your design be different if you were not using this database? How many iterations did your schema design require? Also mention any other design ideas you had, and why you chose this one? Answers backed by evidence (actual test results and bar charts) will be valued highly.*

The input query , contains userid and timestamp, while the output query would have the tweet id sentiment score and tweet text. Creating different columns for each of these fields in the database would have been expensive in terms of querying the database , so the team decided to create a schema with columns as follows :

Column 1:userid# timestamp
Column 2: tweet id : sentiment score : tweet text.

No the schema would still remain same , as the output of mapper and reducer is in format of key-value pair and thus the same format would need to be followed while submitting the data.

2. *What was the most expensive operation / biggest problem with your DB that you had to resolve for Q2? Why does this problem exist in this DB? How did you resolve it? Plot a chart showing the improvements with time.*

Answer:

The most expensive operation is when data is not found in any of the caches and has to be read from the disk. This is because I/O speeds are significantly slower than operations within internal memories of the system.

We solved this problem by employing a dual caching technique: one within the java application and the other at the database level. Data in the java application was cached by using a LinkedHashMap data structure, while in the HBase table, a second level of caching was done by scanning tables and using the setCaching() function.

The graph in question has been shown and explained in question 4.

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

Answer:

1. 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.
2. 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.
3. 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.

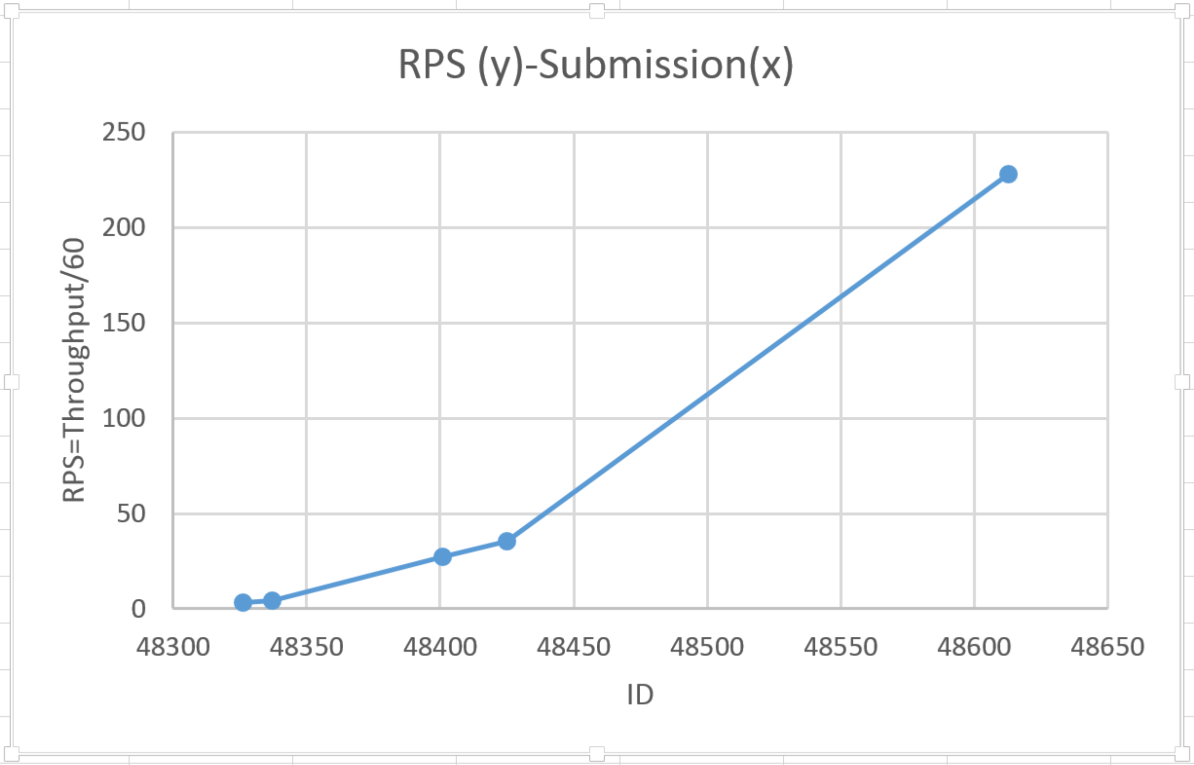
HBase does auto-sharding that is it automatically distributes data across multiple instances when the table size becomes too large. Developers do not need to worry about setting up sharding.

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

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).*

Answer:

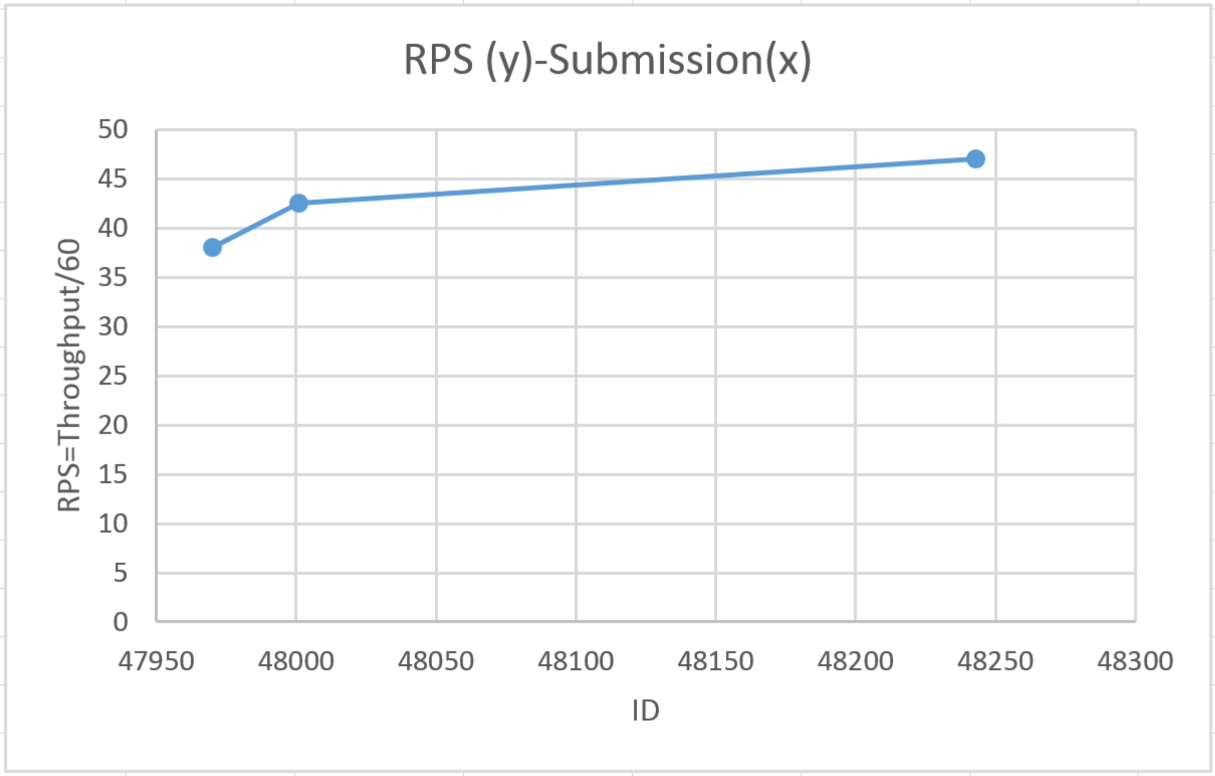
HBase:



Graph: RPS versus ID

At 48326 and 48337, we did not use any kind of optimization. Rather, they were the first runs where we were able to attain correctness. They made us realise about the extent of optimizations required. We realized that we were creating new references to the HBase tables repeatedly in the code. Once we solved that issue, the RPS rose significantly as is represented by submissions 48401 and 48425. 48613 was the first run after we implemented two levels of cache: within the Java code and in Hbase.

MySQL:



Graph: RPS versus ID

All MySQL runs were conducted after indexing the table so the difference was not significant. Changing the cache size didn't yield any significant difference either. The difference between runs 47970, 48001 and 48243 was mainly due to warm-ups.

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

Answer:

Yes, the design would work even for insert operations. For MySQL, we used a single instance which held the java front-end application as well as the database, so inserting could be done by simply adding a method in the code. However, we would need sufficiently large storage to account for the increase in the amount of data stored.

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

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

Answer:

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.

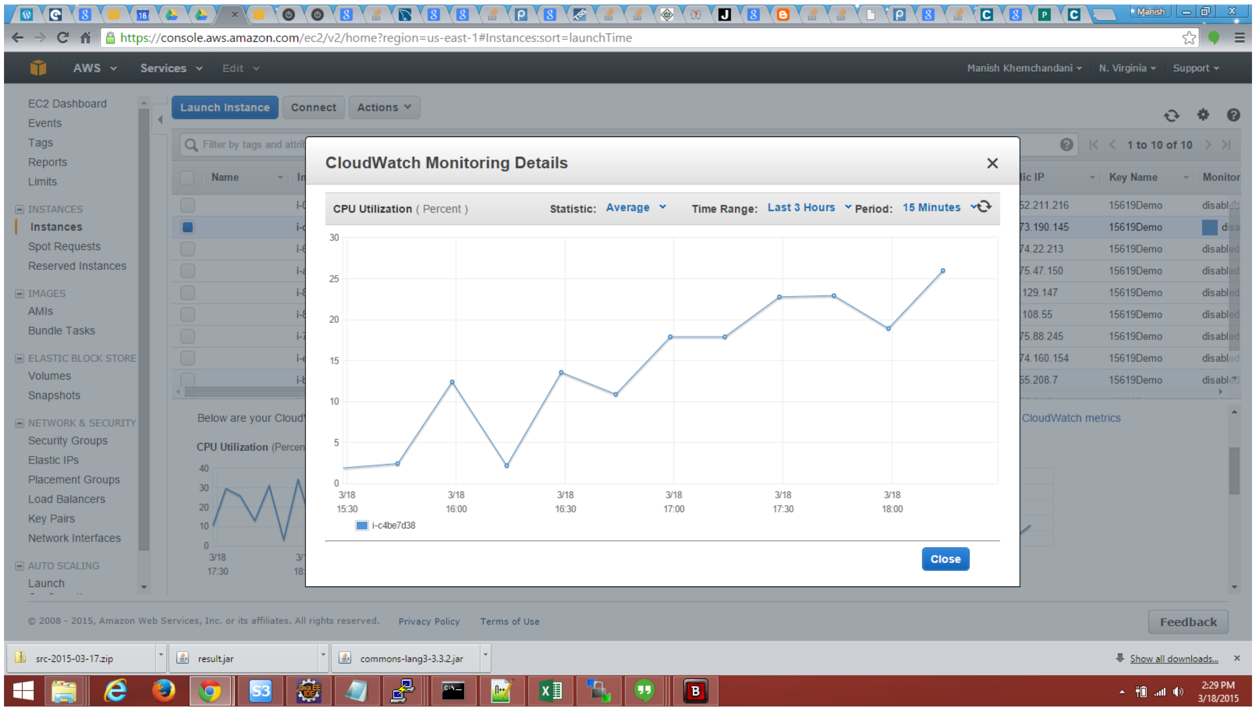
We explored other drivers for connecting to MySQL database, however based upon the performance offered by MySQL connector, we decided to go with it.

7. *How did you profile the backend? If not, why not? Given a typical request-response for each query (q1-q2) what percentage of the overall latency is due to:*

a. *Load Generator to Load Balancer (if any, else merge with b.)*
b. *Load Balancer to Web Service*
c. *Parsing request*
d. *Web Service to DB*
e. *At DB (execution)*
f. *DB to Web Service*
g. *Parsing DB response*
h. *Web Service to LB*
i. *LB to LG*

Answer:

We profiled some aspects of the back-end. Specifically, we captured the CPU utilization of the HBase master node to see whether the CPU was not being over-utilized as the rps count increased.

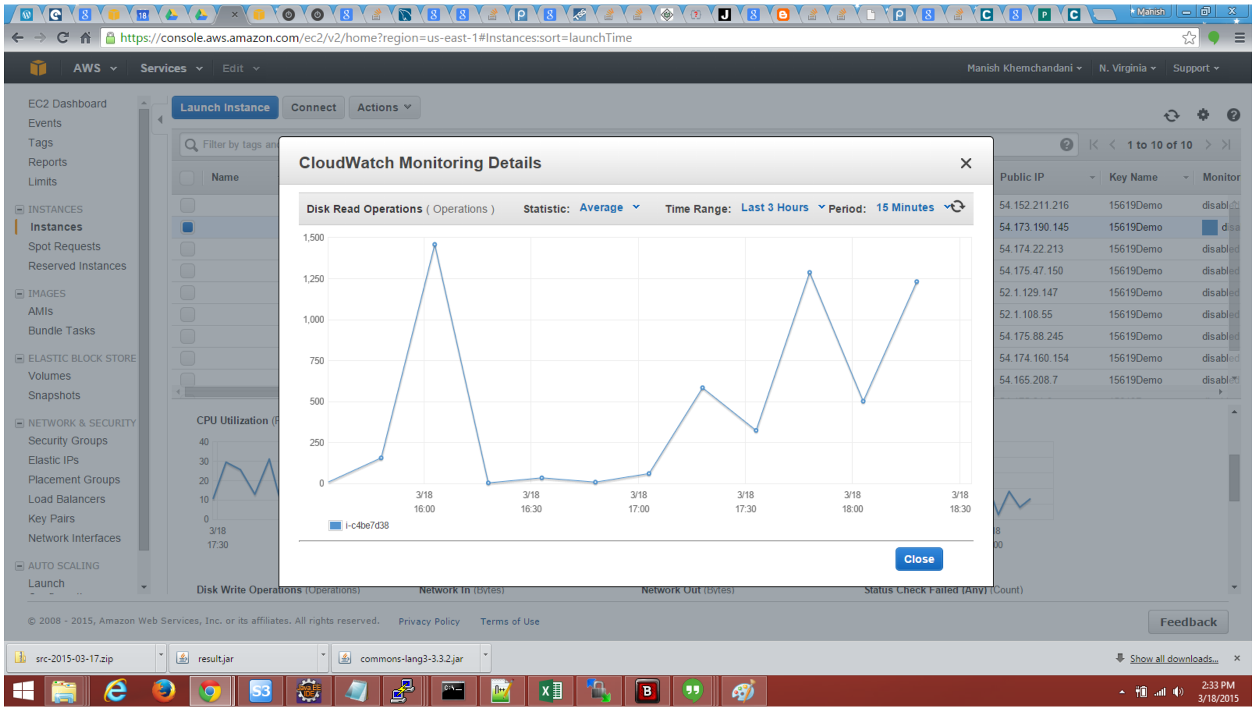


CloudWatch Monitoring Details

CPU Utilization (Percent) Statistic: Average Time Range: Last 3 Hours Period: 15 Minutes

Graph 4: Cpu utilization of HBase

Read operations were also captured to verify the veracity of the rps count on the scoreboard.



https://console.aws.amazon.com/ec2/v2/home?region=us-east-1#Instances:sort=launchTime

CloudWatch Monitoring Details

Disk Read Operations (Operations) Statistic: Average Time Range: Last 3 Hours Period: 15 Minutes

Public IP	Key Name	Monitor
54.152.211.216	15619Demo	disabl
54.173.190.145	15619Demo	disa
54.174.22.213	15619Demo	disabl
54.175.47.150	15619Demo	disabl
52.1.129.147	15619Demo	disabl
52.1.108.55	15619Demo	disabl
54.175.88.245	15619Demo	disabl
54.174.160.154	15619Demo	disabl
54.165.208.7	15619Demo	disabl

Graph5 : Cpu utilization of HBase(read)

8. *Say you are at any big tech company (Google/Facebook/Twitter/Amazon etc.). List one concrete example of an application/query where they should be using NoSQL versus one where they should be using an RDBMS. Both examples should be based on the same company (you choose).*

Answer:

1. Hierarchical data to be stored: NOSQL performs better when given hierarchical data compared to the RDBMS, as it follows the key-value pair way of storing data similar to JSON data. NoSQL database are highly preferred for large data set (i.e for big data). Hbase is an example for this purpose.
2. For scalability: In most typical situations, SQL databases are vertically scalable. You can manage increasing load by increasing the CPU, RAM, SSD, etc., on a single server. On the other hand, NoSQL databases are horizontally scalable. You can just add few more servers easily in your NoSQL database infrastructure to handle the large traffic.
3. For properties: SQL databases emphasizes on ACID properties (Atomicity, Consistency, Isolation and Durability) whereas the NoSQL database follows the Brewers CAP theorem (Consistency, Availability and Partition tolerance)

9. *What was the cost to develop your back end system?*
The total cost for Query 2 was 25$ approximately.

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

Answer:

1. http://www.thegeekstuff.com/2014/01/sql-vs-nosql-db/
2. https://www.devbridge.com/articles/benefits-nosql/
3. http://www.tutorialspoint.com/jdbc/jdbc-statements.htm/

Task 3: ETL

1. *For each query, write about:*

a. The programming model used for the ETL job and justification

Answer: Mapper- Reducer was used for ETL job. Twitter data was picked up by Mapper and the operation for sentiments score calculation and text censoring was implemented based on the criteria provided in the writeup. 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.

b. The number and type of instances used and justification

Answer: The type of instances that the team used were m3.xlarge. Though the instances were large, after confirming on Piazza , team tried to calculate the time cost tradeoff with m3.large and m3.xlarge. The team found out that m3.xlarge had a difference of cost * while m1.large has *.

c. The spot cost for all instances used

Answer**:** m3.xlarge- 0.04 (approximately)

d. The execution time for the entire ETL process

Answer: The execution time for entire ETL process was around 50 minutes by using m3.xlarge .The output were 39 part files, with a total size of 50 gb.

*e. The overall cost of the ETL process***:**

Answer: Final Run Cost: 1.34

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

Answer: 5 runs starting from testing on 1 file to going up to 20 files.

g. Discuss difficulties encountered:

Answer:

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.

*h. The size of the resulting database and reasoning***:**

Answer: The resulting rows were around 269600000.

i. The size of the backup:

Answer: The team thinks that the size of the file obtained by the EMR job is considered to be the size of the backup file, which was around 50 Gb.

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

Answer - The team tried the following techniques to make the ETL job more time efficient.

The team used larger instances (m3.xlarge) which reduce the processing time to just 50 minutes and the price was at the lower side too (approximately 1 dollars).

-The team also tried to reduce the data horizontally by combining on data which had same userID and Timestamp.

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

Answer: As discussed in the above answers the team did use EMR for running the ETL jobs. As it helps in handling large amount of data and outputs data in sorted format. The team didn't consider any of the 2 approaches, and decided to go with the default settings. As far as the knowledge of team the streaming EMR would be highly useful had we considered going with any language other than Java.

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

Answer: The team used s3 explorer to download the files generated (the primer video on configuring and downloading data saved our lives) by the EMR and which were saved in the s3Bucket.

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

Answer: The team feels that MySQL was easier to configure and use when compared to HBase. Since the team used only one instance downloading MySQL server and loading the data became even easier. Also there were no substantial amount of setups and steps required to configure the mySQL . Given the team members had quite a good amount of experience in mySQL and with the available resources available, it was easier to use mySQL. Since the limited experience and resources on Hbase , the team had to face many issues while setting up Hbase, though it helped in achieving the best rps (meaning all our hard work and piazza post scanning on HBase issues didn't go in vain)

[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?*

Answer: Our design for phase 1 will work for a large database, though the team thinks that the rps would go low for both MySQL and HBase , and thus would need more warm ups, use of horizontal scaling and change in caching values , in order to achieve a better performance.

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

No we didn’t attempt to generate load on our own. The team decided to check the functionality by running individual queries , as we figured out that creating a load generator on our own would take up time and thus using a prebuilt load generator was a better option.

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

Answer:

For phase 1 query 1 , the team wished to try undertow framework , but since lack of time we only got time to use this framework towards the second Query and thus required a little rework.

For phase 1 query 2: Given the time constraint the team was unable to resolve the issues with the implementation of ELB for mySQL job. Caching in Backend, Frontend and indexing on tables helped us in attaining a score of 50.Yet, if we had more time , we would certainly have implemented the ELB part and played around with it to generate more rps(and a better score) for mySQL part as well. The team also missed on opportunities to implement sharding for mySQL and consider the rps, or to index the HBase and check the rps, which might have helped us in achieving a better rps altogether.

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

Answer: Phase 1

Query 1

Finding Framework: Kicking off and the team work and deciding on the framework which would be easy to learn and implement was one of the roadblocks in the Phase 1. Lack of documentation on most of the frameworks led us to revert back and use our experience in servlets to implement the query 1. It was not only time taking by running it , but the team very well knew it would have to rework for phase 2 as the obtaining the required rps for the query 2 would be highly infeasible with the servlets.

Query 2:

Logical errors: In Query 2 the team took a lot of time in figuring out how to distribute data on mySQL clusters. Though the team had gained background on this from Project 3.2 , it still didn’t work out for us in this project. After spending around 2 days , the team decided to move to create one instance which had a size of 150 gib and would have all the data. First we created a node by attaching EBS volume to it, which took a 7 hour run to create an index on database. After creating the index, the team realized that it had missed on evaluating certain special characters while loading the data to the table. This mistake led the team to lose another 9 hours , out of which 2 hours were spent on creating new tables and modifying the load commands which used “optionally escaped by “\\”” part inside the command. The team had also to work on modification of code part where it used escapeJava and unescapeJava functions of String escapeUtils methods of Apache common packages. The team also missed on the end of line termination delimiter, because of which most of the submissions resulted in high error rate .

Team being Unlucky : One of the unlucky period was when the team members ran the indexing part and nicely slept thinking that everything would work fine and the new morning will bring all the sunshine . But God had created a different part for us, and had set this path to be to adventurous. When we opened eyes, and looked at the screen , the message said that instances had been terminated. The whole world shattered for us, 1 week of hard work had gone in vain. But putting up the strength and listing 4 times the price of spot instances we re ran the load, and concurrently started working on Hbase part, and some how everything was in working by 7 p.m. and by 11:50 we had reached to the expected (more than expected) score .

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

Answer: It is often said that sometimes the best outcomes are produced under pressure. The team understanding that attaching the EBS volume was the factor that was leading the indexing part to take such high amount of time. The team ran two instances, one with EBS volume (7 hours) and the other without EBS volume (1 hour and 11 minutes). This helped us in saving the time on indexing part. Also in the HBASE part the team used the scan class which allows to scan the database and implement the auto caching. Implement this caching was easy and did help in raising the rps to a high extent. Using prepared statement in mySQL helped in increasing rps , as it stores the pre compiled queries and thus in terms of large queries it helps in creating a significant improvement in terms of rps.