15619 Project Phase 1 Report

Performance Data and Configurations

	Front end	Web service with HBase	Web service with MySQL
Query	q1	q2 (small dataset)	q2 (small dataset)
Scoreboard request ID	6863	-	8714
Instance type	m3.large	-	m3.large
Number of instances	4	-	4
Queries Per Second (QPS)	11065.9	-	5827.9
Error rate	0.00	-	0.00
Correctness	100.0	-	72.0
Cost per hour	0.585	-	0.585

Task 1: Front end

Questions
1. Which front end system solution did you use? Explain why did you decide to use this solution.

We use Apache2 + PHP as our solution. The main reason we choose PHP is because we don't have any experience with web developing, so we need a general solution that most people apply. Therefore, we can easily get well-documented online resource. However, the performance of this solution is not so good, so we might decide to use another solution such as Tomcat in next phase.

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

Our performance might be limited by Apache2 and PHP, so we can not achieve the throughput requirement. Therefore, we decide to use better instances to compensate for our software, so we choose to use m3.large instances. Moreover, considering that bigger is not always better for ELB, we decide to use only 4 instances after evaluating the cost and performance ratio in series of testing.

3. Did you do any special configurations on your front end system? Explain your design decisions and provide details here.

On our front end system, we install Apache2 and PHP5. In order to fix the issue that Apache2 does not understand requests without extension name (such as q1 instead of q1.php), we must put a .htaccess file to fix it. In this .htaccess file, we tell the Apache2 that it should try its best matching requests without extension name to those existed PHP file (Options +MultiViews AddType application/x-httpd-php .php).

4. What is the cost to develop the front end system.

We tried to improve the performance, so we code our own service with socket programming to response the HTTP request in C. Even using only one single m3.large instance could get a pretty nice 8000+ throughput. Unfortunately, we failed to comply with ELB, so we didn't apply our C service. Therefore, we wasted some budget on this trial and error part. The cost of the development might be over $3 including lots of ELB and spot instance (over 50 m3.medium instance-hour) in variety of types.

Task 2: Back end (database)

Questions
1. Describe your table design for both HBase and MySQL. Explain your design decision.

For MySQL, we decided to include 5 columns which are userid, time, tweetid, sentiment, and censored text. Since the queries are on time and userid, we decided to index on time and userid. We designed our database this way to take advantage of the indexed userid and time. However, we realized we will have even faster queries if we have userid and time in one columns, so we will be utilizing that for the next phase.

For HBase, because keys need to be unique, we decided on appending time to userid along with an extra number. That way when we query, we can query on keys that match <userid>_<time>_* to return multiple rows. For our columns, similar to mysql, we had tweetid, sentiment and censored text. Because of budget constraints and difficulties using HBase, we were unable to finish this component.

2. What is the cost to develop your back end system.

Not including the cost of running etl to produce the output needed, we spent approximately $7 to develop our back system. Also, we weren't able to completely complete our hbase backend, which also affected our costs.

Task 3: ETL
Since ETL was performed for both HBase and MySQL, you will be required to submit

information for each type of database.

<u>MySQL</u>:

1. The code for the ETL job

   In the directory ETL including: Q2Mapper.py, Q2Reducer.py, and translaterot13.py

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

   The map job extracted the needed values from a json tweet (userid, time, tweetid, text) and sorted them by userid. The reduce job did sentiment analysis and censored the text, outputting them to a comma separated text file.

3. The type of instances used and justification

   We used m3.x2large in order to speed up the processing of reducer. The lesson we learned was that we should not use python for mapper and reducer, since it's very slow.

4. The number of instances used and justification

   We used 1 master and 19 core instances in our MapReduce configuration, since we needed to speed up our processing of python reducer. In order to solve the problem, we have decided to replace python with Java in next phase.

5. The spot cost for all instances used

   We checked the history, and found that the history of the price of m3.x2large was stable ($0.0645 per hour in usual, but burst up to $3 sometimes). Since our jobs would last a long time (around 12 hours) and we didn't want our jobs to be interrupted, we bid our spot instance with the highest price $3.6. Fortunately, we got only one burst price $0.9 during our ETL. So the price is roughly
   0.0645 * 20(number of instances) * 12(hours) + 0.9(burst price) * 20 *1 ~= $33.48
   By checking the bill, our spot cost is $40, close to our estimation.

6. The execution time for the entire ETL process

   We spent 13 hours to do the extract part by applying EMR and got 125 output files. After that, we created an EC2 instance and used s3cmd to download the output from reducers. It took less than an hour (totally ~30GB output, and download speed ~45MB/s) to transport the output. After that, we loaded these output into mysql. It took less than an hour to load these output into table, but it took almost an hour to create index. To sum up, the execution time for the entire ETL process is around 16 hours.

7. The overall cost of the ETL process

   ~$10 for medium run. However, under the pressure of the deadline, we spend ~$40 for x2large run.

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

   There were 2 incomplete ETL runs before our final run. One was because of a trivial bug in reducer that we didn't find on our small dataset test. Another was because our bid price on spot instances was low so that they got terminated during our ETL process.

9. Discuss difficulties encountered

   python being too slow, newline, comma, special characters, unicode, spot instances terminating halfway.

10. The size of the resulting database and reasoning

    The size of the database took almost 35 GB. Since the size of original source data was almost 30GB (see Question 12), it's reasonable that the size of database was a little bigger than that since the database needed to store some extra information for indexing and data structure.

11. The time required to backup the database on S3

    We didn't backup the database on S3. Instead, we created a snapshot after loading all outputs into mysql and indexing them. The time creating the image was around 20 minutes.

12. The size of S3 backup

    The output of reducer were 27.72GB (29766120530 bytes) totally.

<u>HBase</u>:

The Extract part of HBase is the same with MySQL. However, we didn't finish the loading part, because we encounter a problem trying to import the csv data from reducer into HBase database. Although we have finished our deploying java code, we didn't know how to deploying custom jar in EMR. Therefore, we might have no answer for some questions in this part.

13. The code for the ETL job
    Q2Mapper.py, Q2Reducer.py, translaterot13.py, HBaseImport.java

14. The programming model used for the ETL job and justification
    We used the results from ETL on mysql and java to create insertable code for HBase.

15. The type of instances used and justification
16. The number of instances used and justification
17. The spot cost for all instances used
18. The execution time for the entire ETL process
19. The overall cost of the ETL process
20. The number of incomplete ETL runs before your final run
21. Discuss difficulties encountered
    Loading step: Import data into HBase.

22. The size of the resulting database and reasoning
23. The time required to backup the database on S3
24. The size of S3 backup

Questions

1. Describe a MySQL database and typical use cases.
   MySQL is an open-source relational database that uses Structured Query Language (SQL). A MySQL database consists of tables. Each table contain records (the rows), and each of the records contain fields (the columns)

   Use cases:
   Typical use cases include databases which can can be effectively queried with slower write/read ratios and where scaling massively is not a major concern. From our experiences in our project, we could definitely see that MySQL was way more easier to install, use and query data - hence it would be ideally suited for use cases where ease of use is a priority (over speed perhaps?).

   Also, since it’s open-source, companies/applications that face tight budgets could use this over other types of databases. Since, the cost of usage is lower than other databases, due to free usage and single non-distributed server/database instance, it is suited for applications on a tight budget.

   Also, it’s hugely popular (at least currently), so finding support for troubleshooting, installation and maintenance is very easy - again, very much suited for cases where starting off from the ground up is more important than speed and efficiency.

2. Describe an HBase database and typical use cases.
   HBase is a column-oriented database management system that runs on top of HDFS

(Hadoop Distributed File System). It is an open-source version of Google's BigTable. It comprises of a set of tables, where each table contains rows and columns, like a traditional database. Each table must have an row element defined as a Primary Key, and all access attempts to HBase tables are done via this primary key. The elements in a HBase database are sorted according to the primary key, which are raw byte arrays. Each key consists of the following parts: rowkey, column_family, column and timestamp. mapped in the following manner :(rowkey, column family, column, timestamp) -> value

Columns are grouped into column families and have names to distinguish between them. A colon character is used to delimit column prefix from a family member. HBase is a columnar database in the sense that all of the column's family members are stored together on the underlying file system.

Use Cases:
HBase is applicable where an application has a variable schema where each row is slightly different. Also if data is stored in collections, HBase's columnar database advantages could be ideally leveraged. Since HBase is run on top of HDFS, it is ideally suited for cases where distributed systems can be procured affordably. It is ideally suited for logging analyses and analytics.

3. What are the advantages and disadvantages of MySQL?
   Advantage
   - Open-source (at least majorly) and cheap to run (no need of a distributed file system)
   - Cross platform support - can be used on any OS/machine.
   - Easy to install and use - so it's perfect to get things started off the ground.
   - Large amount of support available for difficulties, since it has almost been an industry-wide standard.

   Disadvantage:
   - Strongly dependent on hard disk performance since it's a transactional relational database.
   - It have trouble clustering, and scaling horizontally.
   - A few reports that we came across highlighted stability issues, especially for large databases.
   - Not very feature rich like its alternatives like HBASE, PostgreSQL, MongoDB, etc.
   - Some algorithmic aspects lack efficiency and speed, when compared to more recent databases.

4. What are the advantages and disadvantages of HBase?
   Advantages:
   - It's very flexible since new columns can be added to families at any time and

therefore able to adapt to changing application requirements.
- Faster access times, so data can be queried with lower latencies. Also, databases can be loaded quickly because of MapReduce capabilities of Hadoop.
- Highly scalable, so larger datasets can be added without much difficulty.

Disadvantages
- It cannot automatically validate data values when they are loaded into the table.
- It has proved to be notoriously difficult and complex to setup and run, as we've noticed in phase-1. Setting up of the master, zookeeper and cluster nodes and configuring them was a bit cumbersome, when compared to the straightforward approach we had with MySQL.
- The machines have to be kept alive to prevent termination