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	1246	1922	1896
Instance type	m1.large	m1.large	m1.large
Number of instances	1	4	1
Queries Per Second (QPS)	7312.2	1953.5	4242.6
Error rate	0.0	0.0	0.0
Correctness	100.0	100.0	100.0
Cost per hour	***Spot instances:*** **$0.03/hour** *On-demand instances:* $0.24/hour	***Spot instances:*** **$0.12/hour** *On-demand instances:* $0.96/hour	***Spot instances:*** **$0.03/hour** *On-demand instances:* $0.24/hour

Task 1: Front end

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

Initially, we used Java Servlets with the Apache Tomcat web server, based on the TAs' recommendation. We reasoned that this would be highly efficient since we were using bare-metal servlets, with none of the extra maintenance and operational overhead that comes within more sophisticated web frameworks. However, for q1, we only got a maximum throughput of 4170 and minimum latency of 23 with this option (check job request id 1058), using a single m1.large EC2 instance; we were not satisfied with this, so we looked into other options before we moved onto q2 itself.

Ultimately, we used JBoss' **Undertow IO** because according to Techempower.com, Undertow is among the top 2 web frameworks (when running on a m1.large EC2 instance) with regards to performance in the areas of JSON serialization, database queries, and plaintext HTTP responses. Given that query Q1 is testing the server's efficiency with regards to plaintext HTTP responses, and query Q2 is testing the server's efficiency with regards to fetching tweet ids from a database, we felt that Undertow was a good fit for our needs. Undertow's efficiency is primarily due to the fact that it uses non-blocking APIs and leverages Java NIO (which is a collection of libraries built on top of Java sockets that are optimized for intensive I/O operations). At any rate, Techempower was proved right as we managed to get a throughput of 7312.2 and a latency of 12.0 after we used Undertow for query Q1 (check job request id 1246).

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

We used a m1.large instance because it was the instance with highest compute power and storage capacity out of the three types that we were allowed (m1.small, m1.medium, and m1.large). m1.large instances have double the virtual CPUs and double the ECU (a measure of compute capacity) of m1.medium instances and quadruple that of m1.small instances. m1.large also has about double the storage of m1.medium and 5x that of m1.small. Since the tests were aimed to stress/overload our servers, we needed a server that had the computing power to handle a large number of requests as quickly as possible and the storage capacity to store and process a large set of information. And m1.large fit the bill.

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

Originally, when we used Java Servlets and Apache Tomcat, we obtained maximum performance by setting Tomcat's maximum number of threads to 1024 and increasing the JVM heap size to 2048 MB (2 GB). We noticed that after this point, if we increased the thread size any further, performance would degrade as there were too many threads competing with each other for limited resources (thereby increasing thread maintenance overhead). By increasing the JVM heap size to 4096 MB (4 GB), we did manage to increase the maximum number of threads that could be optimally supported, but this got us a negligible performance gain, mostly because we couldn't mitigate the overhead due to the competition between many threads.

Now, in Undertow, we haven't performed any explicit configurations since the defaults are already geared towards optimal performance. However, if we were to do so, we would have used a call to setMaxThreads (a method in Undertow's API) when constructing the server object to set the maximum number of threads to approximately 1000.

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

Front end Development and test = 16 * 0.0131 (medium EC2) + 65 * 0.0071 (small EC2) + 15 * 0.0262 (large EC2) = $1.06

Task 2: Back end (database)

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

MySQL

In our MySQL database, we had a single table called “tweets,” which had 2 columns. One column, which was also designated as the primary key, was “userid_timestamp” and the other column was a ‘_’ delimited set of tweet ids (a list of tweet ids separated by ‘_’).

An alternative database schema that we considered was to use the tweet id as the primary key and then map user id and timestamp to the tweet id. This would have conformed to the principles of a more traditional, normalized database schema. However, we pursued a more denormalized structure instead because this structure allows us to get all the tweets posted by a user at a given time at once, using a single SELECT statement rather than the multiple SELECT statements required by the other approach. Thus, our database schema minimizes the number of database queries need to fetch the list of tweets for a given “userid_timestamp,” thereby maximizing the throughput and minimizing the latency.

HBase

In our Hbase database, we use “userid_timestamp” as our row key, only define one column “tweetID”, and use the column prefix “tweetID” as the key to get the cell value. The cells belong to this column keep all of tweet IDs posted by given userid and at give timestamp as the row key. Similarly, this design come out for performance consideration. In this way, we can use a row key to fetch a target row and get the list of tweet ids directly by a given static key “tweetID”.

Also, since Hbase is developed for big data, it is very common to store web page or article content in the cell, so it makes sense that our design will store a bunch of tweet ids with a large size.

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

Total cost = 7 * 0.0131 (medium EC2) + 46 * 0.0071 (small EC2) + 108 * 0.0262 (large EC2) + 108 * 0.06 = 11.63

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

   As submitted.

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

   i. ET(extract and transform) : MapReduce

   Because of our database schema, we do MapReduce job for extract and transform steps. During the Map step, the master node takes the input and divides it into smaller sub-problems, and distributes them to worker nodes. The worker nodes return a list of pairs data (userid_timestamp, tweet_id). Since the data pair with a given key “userid_timestamp” will all sent to a specific worker nodes, during the Reduce step, the reducers aggregate the tweet_id for every given key “userid_timestamp” and write out to separate csv files.

ii. L(load)

Use a script to read the csv files from S3 bucket and load the csv files into MySQL database sequentially.

3. The type of instances used and justification

   m1.medium is used since on the day which we did the ETL, spot price for small instance small jumped to $0.5 for no reason and last for extended period of time. (Bug in pricing system suspected) Thus we used the spot medium instances instead. The reason why we did not use another zone is that we had a hbase database up running in the same zone. We did not want to waste the budget invested in the hbase.

4. The number of instances used and justification

   9 instances used. 1 master node and 8 slave nodes were used for the map-reduce job. Infact small instance would be enough, we chose the medium instances instead due to the unexpected spot price for us-east-1b zone.

5. The spot cost for all instances used

   7 * 0.0131 (medium EC2) + 46 * 0.0071 (small EC2) = 0.41

6. The execution time for the entire ETL process

   34 minutes.

7. The overall cost of the ETL process

   Run with ETL process of HBase

   ET (extract and transform) step EMR = 70 * 0.0071 (small EC2) + 18 * 0.0131 (medium EC2) + 70 * 0.015 (small EMR) + 18 * 0.03 (medium EMR) = 2.32

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

   6, including configuration related issues and the interruption because of the sudden jump in small instance. (We had a map-reduce job running at the time it jumped causing the job aborted)

9. Discuss difficulties encountered.

   The design of the database schema is the key point, and the script to access S3 buckets and load the csv files into database is the other issues. Also, the EL (extract and transform) step is also a difficult part.

10. The size of the resulting database and reasoning

    804117 records

11. The time required to backup the database on S3

Actually, we just backup the instance with the MySQL database as AMI image. It takes about 5 min.

12. The size of S3 backup
    AMI size.

<u>HBase</u>:

13. The code for the ETL job
    as submitted.

14. The programming model used for the ETL job and justification
    Same as MySQL. In fact, we extracted data for MySQL and HBase at the same run. The data was extracted from the csv file and processed into the format that we wanted. In the reducer job, we both printed the result out to a file to be used for MySQL data load and put a record into the hbase.

15. The type of instances used and justification
    9 instances used. 1 master node and 8 slave nodes were used for the map-reduce job. Infact small instance would be enough, we chose the medium instances instead due to the unexpected spot price for us-east-1b zone.

16. The spot cost for all instances used
    108 => large instance hour
    108 * 0.0262 (large EC2) + 108 * 0.06 = 9.31

17. The execution time for the entire ETL process
    34 minutes

18. The overall cost of the ETL process
    Run with ETL process of MySQL
    ET (extract and transform) step EMR = 70 * 0.0071 (small EC2) + 18 * 0.0131 (medium EC2) + 70 * 0.015 (small EMR) + 18 * 0.03 (medium EMR) = 2.32

19. The number of incomplete ETL runs before your final run
    6, including configuration related issues and the interruption because of the sudden jump in small instance. (We had a map-reduce job running at the time it jumped causing the job aborted)

20. Discuss difficulties encountered
    The configuration of HBase as provided in the handouts led to a large amount of failed EMR. Documentation and references online are not detailed enough to point out the issue directly.

21. The size of the resulting database and reasoning
    804117 lines in total.

22. The time required to backup the database on S3
    1 minutes.

23. The size of S3 backup
    38.1MB

Questions

1. Describe a MySQL database and typical use cases.

MySQL databases are relational databases, which means that data is modeled as records in tables, where each column represents an attribute/property of a record, and each row represents a record (another way to think of this is that each row clumps together a group of related column fields/attributes). Each row has a primary key, which identifies a unique characteristic of that row and thereby facilitates speedy access to it. Besides the relationship between the rows and the columns within the table itself, MySQL (and other RDBMS) also feature relationships between tables (many-to-one, one-to-many, many-to-many) through foreign keys. Also, note that MySQL runs on top of the local filesystem, like other relational databases.

Some typical use cases for MySQL databases include a Customer Relationship Management database, which stores customers' contact information in various tables, and transactional systems. An example of the latter is a database for a customer's financial transactions at a bank or a store. Overall, MySQL databases are optimal for data that have a rigid, predefined structure/organization which can help with the processing task for which they are needed.

2. Describe an HBase database and typical use cases.

As with RDBMS, applications organize data into rows and columns in tables within an HBase database. However, a row in an HBase database is accessed using a unique row key, which is a raw array of bytes. This means that we can technically use any serialized/serializable object as a row key. Also, HBase maintains a table's rows in sorted order, sorting by the row key. Columns can be grouped into column families and don't have to be typed since they are also processed as raw byte strings; this gives us more flexibility since we can store any type of data in the table.
HBase is best suited for big-data storage, which requires fast access to multiple rows at a time for aggregations, such as summing or averaging.

3. What are the advantages and disadvantages of MySQL?

Advantages
MySQL is fully ACID-compliant and has transactional properties, which provides guarantees about the safe storage and manipulation of data.
Also, MySQL can automatically validate data values when they are loaded into a table since we have to specify a type for each column when we create the table.

Disadvantages
Since MySQL is fully ACID-compliant and was primarily designed to run on a single machine, there are many complications as we scale it up to a distributed system. In a distributed system, MySQL experiences performance degradation as we implement complex protocols (such as the 2PC protocol) to ensure full ACID-compliance.
Also, according to the CAP theorem, a distributed system with shared data can only have at most two of consistency, availability, and partitioning but not all three at any given time. In a high-performance, large-scale distributed system, partition tolerance is mandatory since we are dealing with a large number of nodes, which significantly increases the chance of failure. This forces us to choose between consistency and availability, and we have to either weaken MySQL's consistency guarantees or sacrifice availability. Overall, MySQL does not scale well.

4. What are the advantages and disadvantages of HBase?

Advantages
- Since HBase's columns are not typed, we can store different types of data within the same table; this increases the database's flexibility.
- We also get the flexibility of adding columns to the database at any time.
- HBase's versioning property provides access both to current and previous versions of data.
- HBase was specifically designed to address the scaling issues of RDBMS, so it scales much better than MySQL.

Disadvantages
- HBase is not fully ACID-compliant by design (the tradeoff that we make for better scalability). It guarantees ACID-compliance for a row but not on operations that span several rows. Also, its relaxed consistency model forces the application developer to verify scan results.
- HBase does not support joins.
- Since HBase's columns are not typed, HBase cannot validate data values as they are added to the table.