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	622	1789	2453
Instance type	m1.large	m1.large	m1.large
Number of instances	1	6	2
Queries Per Second (QPS)	4196.1	1270	2485.5
Error rate	0	0	0
Correctness	100	100	100
Cost per hour (on-demand)	$0.24	$1.465	$0.48
Cost breakdown	m1.large ($0.24)	6 m1.large ($0.24 x 6) ELB ($0.025 x 1)	2 m1.large ($0.24 x 2)

Task 1: Front end

Questions

1. Which front end system solution did you use? Explain why did you decide to use this solution.
We used Golang + Nginx for the front end.
At first, we tried to implement it by ruby on rails and Apache with Phusion Passenger. However, we couldn't get good performance for q1. Then we surveyed several front-end, like perl CGI, apache + mod_perl + perl. Finally we figured out that we could get good performance by Golang + Nginx and I adopted it.

2. Explain your choice of instance type and numbers for your front end system.
m1.large and I used 1 instance.

We could pass the required throughput by just single front end.
We tested multi front ends and auto-scale. When every server works well, we could get good performance, over 8,000 qps-- Request ID: 868. But we considered auto-scaling is not practical for this test case. This is because the minimum period of resource measurement is one minute and launching instance requires a few minutes. In addition, we need to warm up the ELB. From this consideration, we found that it was difficult to get approximate score in short test, ten minutes.

3. Did you do any special configurations on your front end system? Explain your design decisions and provide details here.
We implemented front-end cache on the front-end server. This very simple cache mechanism. When front end receives a query, it first checks its internal cache. If cache hits, it replies the content to the client and don't send query to back end database server. If cache misses, it sends query to the server and store the result within the cache. In addition, Nginx uses single process, event driven architecture. Thus, we need to take care about connection between backend server. We implemented connection_pool and every event pull the connection from the pool. We use mutex and avoid race condition problem.

4. What is the cost to develop the front end system.
Total: around $2 ($2.005)
t.micro free
Spot m1.large 61 hours * $0.03 = $1.83
Spot m1.small 15 hours * $0.01 = $0.15
ELB (hours) 1hours * $0.025 = $0.025

Task 2: Back end (database)

Questions

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

MySQL

- user_id(bigint), indexed
- tweet_time(String), indexed
- tweet_id(bigint)

We used bigint for user_id and tweet_id because bigint is 8bits integers and tweet_id is 8bits integer.
We add multiple-column index to user_id and tweet_time in considerance about q2 query.

HBase

- **Row key**: Tweeter's user ID + "|" + Tweet time
- **Column family**: A list of IDs of all tweets, separated by "\n"

Since HBase has no indexing and it works by distributing work to multiple nodes, our experiment of using HBase's "get" with "filter" yielded very poor performance. As a result, we knew that HBase does much better for a simple lookup using a row key. So, we have come up to this table design decision.

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

MySQL

Spot m1.large 2hours * $0.03 = $0.06

HBase

t1.micro	200 hours	Free
Spot m1.large	297 hours	* $0.0261 = $7.7517
Spot m1.xlarge	1 hour	* $0.0521 = $0.0521
Spot m3.large	5 hours	* $0.0430 = $0.215
ELB (hours)	3 hours	* $0.025 = $0.075
ELB (transfer)	2,136,420 requests	= 2.13642 MB estimated (1 request ~ 1 kb)
		* $0.008 per GB = $0.000017 (very little -> skip)
Total for HBase	**= $8.09**	

Task 3: ETL

Since ETL was performed for both HBase and MySQL, you will be required to submit information for each type of database.

MySQL:

1. The code for the ETL job

Please see attached for the code.

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

We implemented E and T by Java and JSON, JSON parse library and loaded record to the MySQL by "load data local infile" command.

[E and T]

At first, we implemented E and T by perl and python. But these they required so long calculation, around 40 minutes. On the other hand, when we tested by Java, it could complete in a minute. Thus we adopted java and we didn't need to use any parallel execution mechanism in this phase.

[L]

We checked MySQL document and it stated that "load data local infile" is 20 times faster than using insert. http://dev.mysql.com/doc/refman/5.5/en/insert-speed.html

Thus we used "load data local infile" method.

3. The type of instances used and justification

m1.large.
We could complete ET in a minute and We used m1.large for the back-end server.

4. The number of instances used and justification
1 instance.
We could complete ET in a minute.

5. The spot cost for all instances used
$0.03

6. The execution time for the entire ETL process
around 1 or 2 minutes

7. The overall cost of the ETL process
$0.03

8. The number of incomplete ETL runs before your final run
5, 6 times

9. Discuss difficulties encountered
I was surprised perl and python is so slow compared with Java.
At first, I used int for twee_id and it was overflowed.

10. The size of the resulting database and reasoning
83.2 MB

```
mysql> SELECT SUM(data_length + index_length )/1024/1024 AS total_db_data_in_MB FROM
information_schema.tables WHERE table_schema like "cloud";
+--------------------+
| total_db_data_in_MB |
+--------------------+
|        83.18750000 |
+--------------------+
1 row in set (0.24 sec)
```

11. The time required to backup the database on S3
Less than 1 minute

12. The size of S3 backup
43301738 bytes

HBase:
1. The code for the ETL job

Please see attached for the code. (HBaseImport.java)

2. The programming model used for the ETL job and justification
Our ETL process implementation is developed using Java. It reads JSON data from the source and extract only required fields (Tweet ID, Tweet time and Tweeter's user ID). Then, we used the extracted information to load (put) into the HBase table. This operation is done line-by-line.

However, we needed to deal with a tweet with duplicate tweet time and tweeter's user ID. We did this by checking for existence of loaded record in each line of data ETL. If the record exists, it will sort all tweets by their IDs then update that record.

Since the dataset is small, we performed ETL operation for HBase locally on its master instance. That is to say, we downloaded the dataset and our Java ETL program on the HBase master instance and executed it.

3. The type of instances used and justification
m1.large
We launched EMR clusters using m1.large instances and we planned to perform EML operation locally on their master instance.

4. The number of instances used and justification
1 instance
Regarding a small dataset, time used for ETL using 1 instance in this phase was acceptable. Plus, this saved our cost. In the future, we plan to use MapReduce instead for a bigger dataset.

5. The spot cost for all instances used
Regarding AWS spot instance price history, an average cost for spot m1.large is $0.0261/hour

6. The execution time for the entire ETL process
27 minutes

7. The overall cost of the ETL process
Since AWS charges usage in an hour basis, the overall cost is $0.0261/hour * 1 hour = $0.0261

8. The number of incomplete ETL runs before your final run
about 10 times

9. Discuss difficulties encountered
We didn't know the format of a record with multiple tweet IDs (tweets with the same user ID and time) until we got the test script. We had to re-import and check the results for several times.

10. The size of the resulting database and reasoning
By using the following command:

```
$ hadoop fs -dus /hbase
hdfs://172.31.33.40:9000/hbase       168029686
```

It gives size of the entire HBase database, which is 168MB.

According to the HBase table's structure, here is the size of data of each row
- **Row key's size**: 29 bytes (e.g. length of "635857878|2014-01-22+12:38:04")
- **tweet_id column's size:** *Mostly* 19 bytes (e.g. length of "425970620178759680\n")
- **HBase's KeyValue data structure size:** Key Length + Value Length + Row Length + Column Family Length + Timestamp + Key type = (4 + 4 + 2 + 1 + 8 + 1) = 20 bytes

Therefore, each row's size is at least 76 bytes. Since there are 800,000 rows. The estimated size is 60.8 MB (there are some rows with an extra big tweet_id column.)

However, the database size is much bigger than the calculated size (by 107.2 MB!). We think that it might be necessary data to run HBase database.

11. The time required to backup the database on S3
1 minute

12. The size of S3 backup
By observing a file named "tweet_id", which is a name of the only column in our HBase table, its size is 38.6 MB

Questions

1. Describe a MySQL database and typical use cases.
MySQL is an open-sourced relational database. Tables are defined by schemas and the data is stored as fixed-length fields. Because of its small size and speed, it is best suited for small or medium Web sites.

2. Describe an HBase database and typical use cases.
HBase is open-sourced, non-relational, distributed columnar database. Data is stored in column order. HBase leverages Hadoop Distributed File System (HDFS), which provides very fast access to a very large dataset because of its distributed nature. Therefore, HBase is suitable for big data applications.

3. What are the advantages and disadvantages of MySQL?

Advantage
Consistency: Constraint on a table makes it possible to handle data in a more robust fashion.
Disadvantage
Data flexibility: Rigid schema is not flexible to alter a table.

4. What are the advantages and disadvantages of HBase?

Advantage

Schema flexibility: It is common to add more columns (qualifiers) on-the-fly. (However, altering column families requires disabling a table.)
Scalability: HBase employs a distributed architecture from leveraging HDFS. Since scalability is a strong advantage of the distributed system, HBase is also very scalable.
Performance: Since HBase works on HDFS, it can take advantage of parallel processing. As a result, its performance can be improved by simple horizontal scaling.

Disadvantage

Complex querying: HBase is not suitable for applications that require complex queries like a relational database. It basically provides filtering if a developer would like to do that. But filtering has very poor performance since HBase will simply scan every row and apply the filter row-by-row, without any help of indexing. If developers were to perform complex queries, they have to design their own secondary index tables.

As a result, HBase developers have to carefully design database schema before using HBase as their database. Since altering table's column families requires the table to be temporarily disabled, which means they have to sacrifice system availability.

According to the project, because looking up the table by using a RowKey is the best way to query HBase table, we concatenated both query fields (tweeter's user ID and tweet time) and use it as a RowKey. Then, we stored an answer (tweet ID) as a column.