Twitter Analytics Web Service
Project Competition
Phase 3
(Due at 11:59PM EDT, Apr 22)

Introduction
In the previous 2 Phases, you have already built a web service consisting of a front end system and two different databases, and also worked on 2 datasets of different sizes. In this phase, instead of working on both MySQL and HBase, you are going to choose one of them, implement 3 new types of queries on it and focus on optimizing the performance of the system. For this phase, you will refine you ETL skills, design more complicated database schemas and also have opportunity to do some creative designs.

Dataset
The dataset for this phase is the same as in Phase 2, which is about 200 GBs containing about 70 million tweets.The tweets are in JSON format. For a more detailed explanation, please refer to the overall project writeup.

The dataset is stored in the following folder on S3:
s3://15619project-dataset-s14/phase2

Query types:

- ★ In Phase 3, we do not have a minimum throughput requirement for each query. Instead, we will provide you with a full score requirement on each individual query when we release Phase 3 testing system. Your performance score will be calculated based on that full score requirement. Please refer to the detail score calculation in the overall project writeup.

Your web service's front end will have to handle the following query types through HTTP GET requests **on port 80**:

1. **Heartbeat (q1)**
   The query asks about the state of the web service. The front end server responds with the project team id, AWS account id and the current timestamp. It is generally used as a heartbeat mechanism, but it could be abused here to test whether your front end system can handle varying loads.
   - REST format: GET /q1
     - Example: http://webservice_public_dns/q1
   - response format: TEAMID,AWS_ACCOUNT_ID
     yyyy-MM-dd HH:mm:ss

    - Example: WeCloud,1234-5678-9012
      2013-10-02 00:00:00
- Full score throughput: **12000 qps**
- Full score throughput for mix-q1: **4000 qps**

2. **Find a particular tweet using userid and timestamp (q2)**
   The query asks for the Tweet ID posted by a given user at a specific time. This query can basically test whether your backend database is functioning properly.
   - REST format: GET /q2?userid=uid&tweet_time=timestamp
     - Example: http://webservice_public_dns/q2?userid=123456789&tweet_time=2013-10-02+00:00:00
   - response format: TEAMID,AWS_ACCOUNT_ID
     Tweet ID 1
     Tweet ID 2
     …
     Tweet ID n
     (all Tweet IDs need to be sorted numerically. There should be a line break '\n' at the end of each response.)
     - Example: WeCloud,1234-5678-9012
       12345678987654321
       98765432123456789
   - Full score throughput: **4500 qps**
   - Full score throughput for mix-q2: **800 qps**
   - Correctness file for q2 on small dataset is available at: s3://15619project-dataset-s14/correctness_files/q2s_answers.txt

3. **Who retweeted a tweet (q3)**
   The query asks for the set of userids who have retweeted any tweet posted by a given userid.
   - request format: GET /q3?userid=133710000
     - Example: http://webservice_public_dns/q3?userid=133710000
   - response format: TEAMID,AWS_ACCOUNT_ID
     userid1
     userid2
     …
     useridn
     (all userids need to be sorted numerically. There should be a line break '\n' at the end of each response.)
     - Example: WeCloud,1234-5678-9012
       1000205192
       1117007780
   - Full score throughput: **7000 qps.**

- Full score throughput for mix-q3: **1000 qps**
- Correctness file for q3 on small dataset is available at: s3://15619project-dataset-s14/correctness_files/q3s_answers.txt

4. **Text of tweets (q4)**

The query asks for the tweetid and text field of tweets created at a specific second.

- REST format: GET /q4?time=2013-10-02+00:00:00
  - Example: http://webservice_public_dns/q4?time=2013-10-02+00:00:00
- response format: TEAMID,AWS_ACCOUNT_ID
  tweetid1:tweet_text1
  tweetid2:tweet_text2
  …
  (all tweets need to be sorted by tweetid numerically. There should be a line break '\n' at the end of each response.)
  - Example: WeCloud,1234-5678-9012
    99963645053419520 0:Happy New Year!
    99963645053421158 4:Merry Christmas!
    ...
- Please note that since the tweet texts in the JSON objects are encoded with unicode, different libraries might parse the text slightly differently. To ensure your correctness, we recommend that you use the following libraries to parse your data.
  - If you are using Java, please use simple json/gson
  - If you are using Python, use json module in standard library
- Full score throughput: **2400 qps**
- Full score throughput for mix-q4: **400 qps**
- Correctness file for q4 on small dataset is available at: s3://15619project-dataset-s14/correctness_files/q4s_answers.txt

5. **Find all the tweets by location and during a particular time range (q5)**

The query asks for all the tweetids that were created during a given time range and the text field of which contains given place.

1. The place argument comes from the *name* field in the *place* object of each tweet in the dataset. It contains only English characters and whitespace. The place will be no more than 3 words and contains at least 2 letters. For example, we will send queries with place=New+Jersey, but we will NOT send queries with place=Montréal or place=5000+Forbes+Ave+Pittsburgh.
2. Sub string does not count for a match, for example, "Pitt" is not a match for "Pittsburgh".
3. Please also notice that the place is case sensitive, and the start_time and end_time are inclusive.

- REST format: GET /q5?start_time=start_timestamp&end_time=end_timestamp&place=some_place

    - Example: http://webservice_public_dns/q5?start_time=2013-10-02+00:00:00&end_time=2013-10-12+11:11:11&place=Carnegie+Mellon+University
- response format: TEAMID,AWS_ACCOUNT_ID
  Tweet ID 1
  Tweet ID 2
  …
  Tweet ID n
  (all Tweet IDs need to be sorted numerically. There should be a line break '\n' at the end of each response.)
    - Example: WeCloud,1234-5678-9012
      12345678987654321
      98765432123456789
      …
- You should be able to handle some invalid times formats, such as 2013-01-32+25:00:00
- Full score throughput: **5000 qps**
- Full score throughput for mix-q4: **1500 qps**
- Correctness file for q5 on small dataset is available at: s3://15619project-dataset-s14/correctness_files/q5s_answers.txt

6. **Number of tweets (q6)**
   The query asks for the total number of tweets sent by a range of userids.
    - request format: GET /q6?userid_min=133710000&userid_max=137713771
        - Example: http://webservice_public_dns/q6?userid_min=133710000&userid_max=137713771
    - response format: TEAMID,AWS_ACCOUNT_ID
      number_of_userid
        - Example: WeCloud,1234-5678-9012
          1337
          (There should be a line break at the end of each response)
    - Full score throughput: **3000 qps**
    - Full score throughput for mix-q4: **200 qps**
    - Correctness file for q6 on small dataset is available at: s3://15619project-dataset-s14/correctness_files/q6s_answers.txt

Tasks

- ★ *Make sure you tag all the instances you use in this phase with the key "**15619project**" and value "**phase_3**". Remember the tags are **case-sensitive**.*
- ★ *Our test system is built in availability zone **us-east-1a**. Please refer to this Piazza note to determine which availability zone you should use.*

Task 1: Front End
In this phase, three new queries are added (q4, q5 and q6). Your first task will be to update your front end system to handle the new queries. Since the performance of the front end affects all queries, you should make sure that your front end system does not limit your overall system performance.

Design constraints:

- Execution model: you can use any web framework you want.
- Cost budget: The front end configuration should not cost more than $0.3/hour at light load and no higher than $1/hour at maximum load.
- Spot instances are highly recommended during the development period, otherwise it is very likely that you will exceed the overall budget (see below) for this step and fail the project.

Front end submission requirements:

1. The design of the front end system
2. Provide all code developed for the front end
3. The type of instances used and justification
4. The number of instances used and justification
5. Other configuration parameters used
6. The cost per hour for the front end system
7. The total development cost of the front end system
8. The throughput of your front end for the given workload of q1 queries

Recommendation:
You might want to consider using auto-scaling because there could be fluctuations in the load over the test period. To test your front end system, use the Heartbeat query (q1). It will be wise to ensure that your system can satisfy the minimum throughput requirement of heartbeat requests before you move forward. However, as you design the front end, make sure to account for the cost. Write an automatic script, or make a new AMI to configure the whole front end instance. It can help to rebuild the front end quickly, which may happen several times in the building process. Please terminate your instances when not needed. Save time or your cost will increase higher than the budget and lead to failure.

Task 2: Back end (database)
In this phase, the most important task is to optimize your back end system performance.

Unlike previous phases, the database you are going to use in this task will be either **HBase** or **MySQL**.

This task requires you to build the back end system. It should be able to store whatever data you need to satisfy the query requests. You **MUST** use spot instances for the back end system

development. But you can use on-demand instances for the live test. Normally, the dataset storage system is the most crucial part of a web service. People cannot take the risk of losing any bit of data, so please use on-demand prices to launch spot instances to reduce the probability of instances termination when testing your system. Do this after you are confident that you have completed system development and worked out your bugs.

For the database design, you should take q2~q6 into consideration, which includes the requirement of throughput and cost. We require you to build only one back end system, meaning you need to choose either ***HBase*** or ***MySQL*** as your back end database. You need to consider the design of the table structure for the database. The design of the table (schema) significantly affects the performance of a database. You need to think carefully over your decision according to the experience you gained from Phase 1 and Phase 2. Whichever database you finally choose, there are trade offs, as different query types might favour different databases.

In this task you should also test and make sure that your front end system connects to your back end database and can get responses for queries.

Design constraints:

- Storage model and justification: you should use either HBase or MySQL and explain your choice.
- Cost budget: The total cost (front end system + back end system) should not cost more than **$1.6/hour** (on-demand prices). If you are using HBase on an EMR cluster, the cost of EMR can be excluded from the cost, because in a real world setting you can configure your Hadoop cluster without the additional cost EMR.
- Spot instances are highly recommended during the development period, otherwise it is very likely that you will exceed this phase's overall budget and fail the project.

Back end submission requirements:

1. The design of the back end system
2. The table structure of the database, justify your design decisions
3. The type of instances (m1.small, m1.medium or m1.large) used and justification
4. The number of instances used and justification
5. The cost per hour for the back end system
6. The spot cost for all instances used
7. The total development cost of the back end system

Recommendations:
Test the functionality of the database with a few dummy entries before the Twitter data is loaded into it. The test ensures that your database can correctly produce the response to the q2 query.

Task 3: ETL

- ★ *The ETL process can last from hours to tens of hours depending on your ETL solution, code efficiency and the database you are using. Testing will be crucial to limit your*

expenditure and improve your debugging process.

★ *ETL is probably the most costly task in the project and you need to seriously consider the approach you use for each query type. We strongly recommend you to consider:*
1. generating some intermediate results and then load into the database
2. performing ETL on multiple queries in the same run to improve efficiency.

This task requires you to load the Twitter dataset into the database using the **extract**, **transform** and **load** (ETL) process in data warehousing. In the extract step, you will extract data from an outside source. For this phase, the outside source is a 200 GB Twitter dataset of JSON file stored on **S3,** containing about 70 million tweets. The transform step applies a series of functions on the extracted data to realize the data needed in the target database. The extract step relies on the schema design of the target database. The load phase loads the data into the target database.

You will have to carefully design the ETL process using AWS resources. Considerations include the programming model used for the ETL job, the type and number of instances needed to execute the job. Given the above, you should be able to come up with an expected time to complete the job and hence an expected overall cost of the ETL job.

Once this step is completed, you **MUST** backup your database to save cost. If you use EMR, you can do the Hbase backup on S3 using the command:

```
./elastic-mapreduce --jobflow j-EMR_Job_Flow --hbase-backup
--backup-dir s3://myawsbucket/backups/j-EMR_Job_Flow
```

The backup will run as a Hadoop MapReduce job and you can monitor it from both Amazon EMR debug tool or by accessing the http://jobtracker-ip:9100. To learn more about Hbase S3 backup, please refer to the following link
http://docs.aws.amazon.com/ElasticMapReduce/latest/DeveloperGuide/emr-hbase-backup-restore.html

Design constraints:

- Programming model: you can use any programming model you see fit for this job.
- AWS resources: You **MUST** use SPOT instances for this step otherwise it is very likely that you will exceed the budget and fail the project.
- ETL code verification: Before you start the ETL process on the large dataset, you **MUST** verify the correctness of your ETL code using the small dataset and corresponding correctness files we provided for each query.

ETL submission requirements for both MySQL and HBase:

1. The code for the ETL job
2. The programming model used for the ETL job and justification
3. The type of instances used and justification
4. The number of instances used and justification

5. The spot cost for all instances used
6. The execution time for the entire ETL process
7. The overall cost of the ETL process
8. The number of incomplete ETL runs before your final run
9. Discuss difficulties encountered
10. The size of the resulting database and reasoning
11. The time required to backup the database on S3
12. The size of S3 backup

Recommendations: For this phase, the ETL jobs will utilize a large dataset (200 GB). Always test the correctness of your system using a small dataset. If your ETL job fails or produces wrong results, you will be burning through your budget and wasting time. You only have 2 weeks for this phase. The ETL job depends on you completing Step 2 (especially for the T and L in ETL). After the database is populated with data, it will be wise to test the throughput of your back end system for different types of queries. Always test that your system produces correct query responses for all query types.

Task 4: Performance Optimization

By this stage, your system is functionally operational and you have already tested its performance and understand its limitations. You might want to consider certain optimizations to performance while accounting for cost. Remember, you are competing against other companies (teams) to win the contract.

Performance Optimization submission requirements:

1. Perform bottleneck analysis in your system.
2. The optimizations utilized based on the analysis. Always provide justifications.
3. Changes to the overall system design for optimization.
4. Changes to the overall system implementation for optimization.
5. For the given workload
    a. The maximum throughput of the optimized system for q1~q6
    b. The latency of your optimized system for q1~q6
6. The cost per hour of your optimized system

Budget

You will have a budget **$75/team** for all the tasks and live test in this phase. Again, make sure you tag all of your instances in this phase with key "***15619project***" and value "***phase_3***". You decide how to spend your budget.

Recommendations:

1. Use smaller instances when you do function development and verification of the front end (you may even finish a lot of the development needed for this task on your own laptop).
2. Think carefully about your database schema to avoid running long ETL jobs too many times.

Deliverables
You need to submit your system for a live test at the end of this phase as in last phase. However, this time **you only need to submit 1 request for live test of your system.** Besides, please submit

1. Project final report in PDF format(Project final report template)
2. All your code written in this phase (front end, ETL, etc...)

To submit your work:

1. Package your completed Project final report and all the source code files you wrote in the phase into a single TEAM_NAME.zip file.
2. Upload the TEAM_NAME.zip file to S3.
3. Submit your S3 link using this Google form

Grading
This phase accounts for 75% of the total grades of the entire project. 50% of the grades is determined by your final live test result, and the other 25% is from your report. Please notice that this is the project final report instead of only for Phase 3. It should document the entire project. You need to incorporate all the information in previous phases into the report.

Live Test Schedule
Final live test is after due of entire project. This live test lasts **4 hours consisting of 8 parts**:
1. 30 minutes warm up test for q1 (not graded)
2. 30 minutes test for q1
3. 30 minutes test for q2
4. 30 minutes test for q3
5. 30 minutes test for q4
6. 30 minutes test for q5
7. 30 minutes test for q6
8. 30 minutes mixed queries test

Hints and References
Front End Suggestions:
Although we do not have any constraints on the front end, the performance of different web frameworks varies a lot. Choosing a bad web framework may have a negative impact on the throughput of every query. Therefore, we strongly recommend that you **do some investigation about this topic** before you start. Techempower provides a very complete benchmark for mainstream web frameworks, you may find it helpful.

You can also compare the performance of different frameworks under our testing environment by testing q1 since it is just a heartbeat message and has no interaction with the back end. Please also think about whether the front end framework you choose has API support for MySQL

and HBase.

General Guideline for ETL:

How to do ETL correctly and efficiently will be a critical part for your success in this project. Notice that ETL on a large dataset could take 10 - 30 hours for just a single run, so it will be very painful to do it more than once, although this may be inevitable since you will be refining your schema throughout your development.

You may find your ETL job extremely time consuming because of the large dataset and poor design of your ETL process. Due to the many reasons that could lead to the failure of your ETL step, please start thinking about your database schema and doing your ETL AS EARLY AS POSSIBLE.

Also try to utilize parallelism as much as possible, loading the data with single process/thread is a waste of time and computing resources, MapReduce may be your friend, though other methods work so long as it can accelerate your ETL process.

Reference documents for MySQL and HBase

MySQL

- http://dev.mysql.com/doc/
- OLI Unit 4 and Project 3

HBase:

- https://hbase.apache.org/
- OLI Unit 4, Module 14
- OLI Project 3 and Project 4