Twitter Analytics Web Service
A 619 Competition

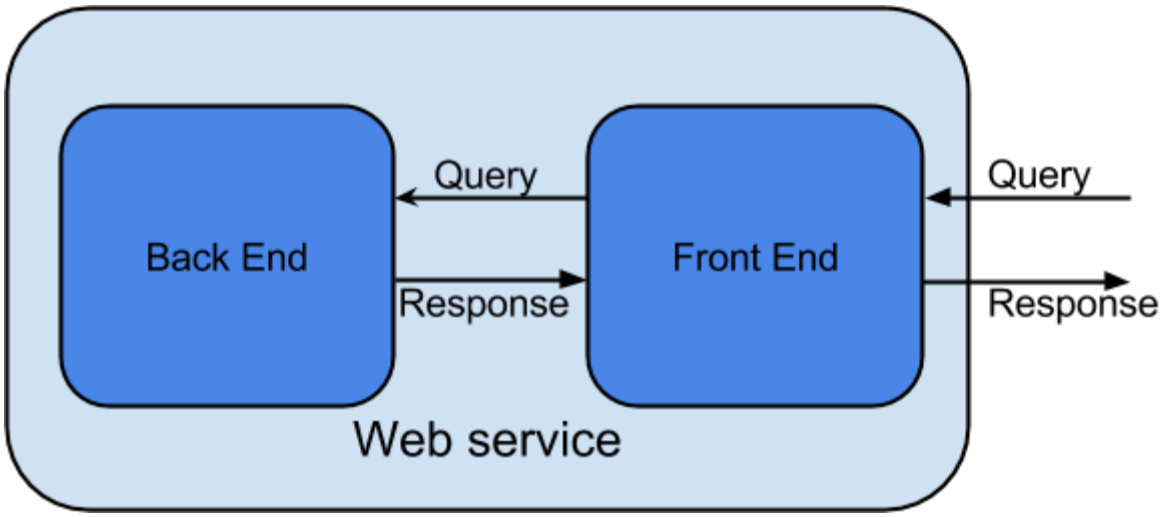
- Introduction
- Timeline
- Web Service Load Generation and Testing System
- Phase 1
- Phase 2
- Phase 3
  - Heartbeat and Authentication (q1)
  - Text Cleaning and Analysis (q2)
  - Retweet Buddies (q3)
  - #WhatsHappeningHere (q4)
  - HotOrNot (q5)
  - Shutter Count (q6)
- Tasks
  - Task 1: Front End
  - Task 2: ETL
  - Task 3: Back end (database)
- Deliverables
  - Phase 1
  - Phase 2
  - Phase 3

Introduction

After making a huge profit for the Massive Surveillance Bureau, you realize that your Cloud Computing skills are being undervalued. To maximize your own profit, you launch a cloud-based startup company with one or two colleagues from 15619.

A client has approached your company and several other companies to compete on a project to build a web service for Twitter data analysis. The client has provided hundreds of gigabytes of raw tweets for your consumption. Your responsibility would be to design, develop and deploy a web service that meets the throughput, budget and query requirements of the client.

Your team needs to build (and optimize) two components:



1. **A Front end:** This should be a web service able to receive and respond to queries. Specifically, the service should handle incoming HTTP requests and provide suitable responses (as defined in the **Query Types** section below). The interface of your service is REST-based.
    a. Users access your service using an HTTP GET request through an endpoint URL. Different URLs are needed for each query type, which are shown in **Query Types** section. Query parameters are included within the URL string of the HTTP request.
    b. An appropriate response should be formulated for each type of query. The response format should be followed exactly otherwise your web service will not provide acceptable responses when tested with our load generator and testing system.
    c. The web service should run smoothly for the entire test period, which lasts for several hours.
    d. The web service must not refuse queries and should tolerate a heavy load.

2. **A Back End:** This is used to store the data to be queried.
    a. You will evaluate SQL (MySQL) and NoSQL (HBase) databases in the first two phases of this project. You should compare their performance for different query types for different dataset sizes. You can then decide on an appropriate storage back end for your final system to compete against other systems.

3. Your web service should meet the requirements for throughput and cost for queries at a provided workload.

4. The overall service (development and deployment test period) should cost under a specified budget. Less is generally better, otherwise your competitor will win the contract.

5. Your client has a limited budget. So, you can **ONLY** use **m1.small**, **m1.medium**, **m1.large, m3.medium and m3.large** instances for your web service (both your front-end and back-end systems). However, you can use spot instances for batch jobs and development but not for the deployed web service (during the live test period).

Objectives:
The goal of this project is to integrate everything that you have learned from the course (and many, many new things) in designing, developing and deploying a real working cloud-based web solution. We encourage you to discover and utilize whatever tools you find. If the tools do not appear in the design constraints of this handout, you **MUST** discuss them with the professor or TAs before using them.

Dataset
The dataset collected by the client is available on **S3**. For phase 1, you should use the folder:

Input and output
The web service solution should provide responses to specific queries on the twitter dataset. Users can submit queries about tweets based on userids, tweet id, tweet time, location, hashtags and photos. The client has collected billions of tweets through Twitter's streaming API in JSON formatted files, which are stored on S3.

The input is in a JSON format. Each line is a JSON object representing a tweet. Twitter's documentation has a good description of the data format for tweets and related entities in this format. https://dev.twitter.com/docs/platform-objects/tweets.

Timeline

This project has 3 phases. In each phase you are required to implement your web service to provide responses to some particular types of queries. The query types will be described in the writeup of each phase.
At the end of each project phase, you need to submit some deliverables including:
1. Performance data of your web service.
2. Cost analysis of the phase
3. All the code related to the phase
4. Answer to questions associated with the the phase.
5. A report for each phase.

The report must include detailed reasoning for your design decisions and deliverables for each phase.

The time allotted for the phases is as follows:

Phases	**Duration**	**Query Type**
Phase 1	2 weeks	Q1, Q2
Phase 2	2 weeks	Q1, Q2, Q3, Q4
Phase 2 Live Test	6 hours	Q1, Q2, Q3, Q4, mix-Q1Q2Q3Q4
Phase 3	2 weeks	Q1, Q2, Q3, Q4, Q5, Q6
Phase 3 Live Test	4 hours	Q1, Q2, Q3, Q4, Q5, Q6, MIX-Q1Q2Q3Q4Q5Q6

Web Service Load Generation and Testing System

In this project, your web service interacts with the 15619 project testing system developed by us to test and grade your service. The testing system has a front end at: https://15619project.org

Registration

You should finish team registration by **Saturday of week 0 (Oct 4th, 2014)** . You need to provide a valid andrew id in order to confirm your registration. You will receive an email to log in and set your password in the first week.

Scoreboard

We provide a real-time scoreboard for you to examine how your performance compares to other teams. Your **best submission** for each query will be displayed on the scoreboard.

Submit your request

You can submit a test request at any time. Please use the **public DNS address** of your instance or ELB so that the testing instance can reach your service. We provide different testing period lengths, please think carefully about what you need to verify for each request and choose an appropriate duration.

- How is my request tested?
  When you submit a request, a testing instance will be launched by us. This instance will run a benchmark program for the specified query to the provided address in order to test the performance and correctness of the web service. After the testing period ends the results will be displayed in your submission history table.

Wait List

Since we have a limited amount of resources, if there are many teams submitting requests at the same time, your request may be waitlisted. You can check to see how long you need to wait before your request starts running. Our system will try to add or deduct resources automatically based on the current average waiting time, so don't worry if you see a long waiting time.

- Why can't I submit a request?
  To save cost and prevent repeat submissions, every team can only have **one request running or waiting**. In other words, if anyone in your team submits a request, no one in your team can submit another request until the running job finishes.

- Can I cancel my request if I modify my service and want to re-submit?
  Yes, you can.

Submission History

When your request is running, you can check the submission history page to see the progress of the current request and how many seconds are remaining. After your request finishes, you your results will be displayed. You can also view all your previous submissions. If you have doubts for a specific query, you can take down the submission ID and contact us for more details.

- How to interpret my result?

There are several numbers for each result:

Throughput: average queries per seconds during the testing period

Latency: Average latency for each query issued during the testing period

Error: The error rate. Your message type in HTTP header of your response must be 2xx. Otherwise it will be recognized as an erroneous response and counted in error rate.

Correctness: This column indicates whether your service responds with the correct result. Your response should exactly match our standard response so please read the format requirement of each query carefully.

Score: This is your final score of this request. You score is based on effective throughput. The effective throughput is calculated as follows:

- **Effective Throughput = Throughput * (100 - Error / 100) * (Correctness / 100)**
    - As you can see, error and correctness can severely impact your effective throughput

Live Test

Phases 2 and 3 culminate with a Live Test, where all systems are simultaneously tested. Your grade for both phases is based on your performance in these tests.

Score Calculation

- The score is a weighted sum of your raw score for each query
- Raw Score = Effective Throughput/Target Throughput
    - The Target Throughput will be provided by us

Bug Report

If you encounter any bugs in this system or have any suggestions for improvement, feel free to use this form to help us improve.

Phase 1

Introduction

In the first phase of the 619 project, you will build two web services from scratch. Each web service has to respond to two types of queries, with data fetched from a storage system, which you must design and control. Each web service connects to a different storage system. A query is an input generated by a test system, which requires a fixed response. Grading depends on the accuracy and performance of your web service's response to these queries.

This phase is designed to ensure that you have the required skills to cope with more complicated queries, so please allocate enough time to finish the requirements by the deadline.

Specifically, you will design and develop a front-end system (a highly parallel, concurrent web server) that can attach to either of two back-end systems (HBase and MySQL). In this phase you will be expected to learn about the advantages and disadvantages of each type of back-end database system.

A report must be written for this phase that conforms to a template.

Dataset

The dataset for this phase is a folder containing about 600 GB of JSON-format tweets.
The dataset is at s3://15619f14twittertest2/600GB

Phase 2

Introduction

In the second phase of the 619 project, you will extend your two web services to support more complex queries. Each web service has to respond to four types of queries, with data fetched from a storage system, which you must design and control.

A report must be written for this phase that conforms to a template.

Dataset

The dataset for **Q3 and Q4** is a folder containing about 1 TB of JSON-format tweets.
The dataset is at s3://15619f14twittertest2/1TB
You can **retain the Phase 1 tables for Q2** (i.e. 600 GB source- no need for another ETL job)

Phase 3

In the third phase of the 619 project, you will choose a single web service and extend it to support two more queries. Retain Q1-Q4 from your earlier dataset. Use the 1 TB dataset for Q5 & Q6 (s3://15619f14twittertest2/1TB).

Phase 3 Query types (these override the Phase 1/2 queries):
Your web service's front end will have to handle the following query types through HTTP GET requests **on port 80 [you cannot use any other port]**:

- **Heartbeat and Authentication (q1)**
  The query asks about the state of the web service. The front end server responds with the project team id, AWS account ids 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.

  For each request to q1, the load generator will send a large numeric key. You need to use this as follows to generate the response:
  a. We give you a public "key" (which is the large prime number below):
     - X=687676683235176539649637753447605000297085748381526291845035586985008516705339467263431539122405215 3
  b. The load generator creates a number X*Y, where Y is another (possibly) huge prime number. X*Y is sent to your web service.
  c. You need to divide this number (X*Y) by X to retrieve Y
  d. You then authenticate by sending Y back to the server as a part of your response
  - REST format: GET /q1?key=<large_number XY>
    - Example:

      http://webservice_public_dns/q1?key=2063030049705529618948913260342815000891257245144578875535106760955025550116018401790294617367215645 9
  - response format:
    <THE NUMBER Y>
    TEAMID,AWS_ACCOUNT_ID1,AWS_ACCOUNT_ID2, AWS_ACCOUNT_ID3
    yyyy-MM-dd HH:mm:ss
  - Example (for the URL above):
    3\n
    TeamCoolCloud,1234-0000-0001,1234-0000-0002,1234-0000-0003\n
    2004-08-15 16:23:42\n
  - Minimum throughput requirement: ~~15000~~ 14000 qps

The following public key is fixed for this project:
X=687676683235176539649637753447605000297085748381526291845035586985008516705339467263431539122405215 3

- **Text Cleaning and Analysis (q2)**
  This query asks for the tweet(s) posted by a given user at a specific time. This query tests that your backend database is functioning properly.

  We send you a user id and tweet time, and you need to send us three pieces of data about the relevant tweet(s).
  a. The tweet ID
  b. The censored text
  c. The sentiment score of that tweet

  Each of these are explained in the following paragraphs.

  (Link to Twitter API)
  Step 1: The tweet id can be fetched from the 'id' or the 'id_str' field.
  Step 2: The tweet text can be fetched with the 'text' field. There are two processes that must run on each tweet text. To do them, you need to split the string into separate "words" wherever a non-alphanumeric character is encountered.

  A) Sentiment Score Calculation:
  We provide a list of lowercase English words that have a corresponding "sentiment" score. For each word in the text, convert it to lowercase and check if it has a sentiment score. If it exists, add the corresponding sentiment value to the score for that tweet. Scoring for all tweets starts with a zero.
  This list is from the AFINN dataset.[1]
  AFINN is a list of English words rated for valence with an integer between minus five (negative) and plus five (positive). The words have been manually labeled by Finn Årup Nielsen in 2009-2011. The file is tab-separated.
  DOWNLOAD IT HERE.

  B) Text Censoring
  Since we have innocent little undergrads also doing this assignment, we did not want to expose them to some of the language that exists on Twitter. So we provide a mechanism to sanitize the text. We give you an ROT13ed version of all banned words that should not occur in your output text. [ROT13 is a simple letter-based substitution cipher. **For instance, the first line in the list of banned words is 15619ppgrfg, which means that the first banned "word" is 15619cctest**.
  This list is filtered from last year's students' colorful posts on Piazza near the

[1] Hansen, L. K., Arvidsson, A., Nielsen, F. Å., Colleoni, E., & Etter, M. (2011). Good friends, bad news-affect and virality in twitter. In *Future information technology* (pp. 34-43). Springer Berlin Heidelberg.

end of the 15619 (all these students started working on their project late).[2] Download it here.

Ensure that in both cases, you use our files (links above) and do not download the original, as we have done some cleanup to make it easy to use for your code.

You must do the steps in order, that is, first calculate the sentiment for a word and then (if required) censor it. A word is censored by replacing the inner characters by asterisks (*).

For e.g. assume that the word **cloud** is on our banned list. Consider:

```
I love Cloud compz... cloud TAs are the best... Yinz shld
tell yr frnz: TAKE CLOUD COMPUTING NEXT SEMESTER!!! Awesome.
It's cloudy tonight.
```

When you reply, it should have the format:

```
I love C***d compz... c***d TAs are the best... Yinz shld
tell yr frnz: TAKE C***D COMPUTING NEXT SEMESTER!!! Awesome.
It's cloudy tonight.
```

And it should have a score of +10 (Best = +3, Love = +3 and Awesome = +4)

Notice the case of all the uncensored letters is maintained. Also notice that "cloudy" is uncensored (since it is not on the list of banned words).

- 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

  You can choose to do all these calculations in the ETL (remember that it will drive up the cost and duration of the MapReduce job) or at the end when the query is requested (if you can write fast code).

- REST format: GET /q2?userid=uid&tweet_time=timestamp
    - Example: http://webservice_public_dns/q2?userid=123456789&tweet_time=2004-08-15+16:23:42

[2] No, actually it's mostly from http://www.bannedwordlist.com/. Decrypting and reading this file could be entertaining and informative as long as you do not use what you learn on the course staff

- response format:
  TEAMID,AWS_ACCOUNT_ID1,AWS_ACCOUNT_ID2,AWS_ACCOUNT_ID3\n
  Tweet ID1:Score1:TWEETTEXT1\n
  Tweet ID2:Score2:TWEETTEXT2\n
  …

(Sort numerically by Tweet ID. There should be a line break '\n' at the end of each response.)

  - Example:

WeCloud,1234-5678-9012,1234-5678-9013,1234-5678-9014\n
12345678987654321:2:When I met Mr Mandela it was one of the most memorable days of my life. A truly inspirational human being....\n
12345678987654323:0:....He will live on in my heart forever. R.I.P\n

- Minimum throughput requirement: ~~7000~~ 5000 rps

Sample for 15619f14twitter-parta-aa.gz at s3://15619samples/F14/q2.sample.gz

- **Retweet Buddies (q3)**
  This query asks for information about how popular a particular users' tweets are. We send you a user id (say X) and you need to respond with a list of users (say A, B, C, D) who have retweeted any of the tweets posted by that user. If any of the users (say C and D) have in return been retweeted by X, then place those user ids within parentheses. The ~~tweets~~ user ids should be sorted numerically. [BUG: It said tweets. Correction→ should have said user ids]

  - REST format: GET /q3?userid=uid
    - Example: http://webservice_public_dns/q3?userid=2495192362
  - response format:
    TEAMID,AWS_ACCOUNT_ID1,AWS_ACCOUNT_ID2,AWS_ACCOUNT_ID3
    342213714\n
    (1532300155)\n
    1663875289\n

Minimum throughput requirement: 10000 rps

- **#WhatsHappeningHere (q4)**

  This query asks for the most popular 'n' hashtag(s) from all tweets from a single location on a single day.
  The hashtags can be read from the entities object (not from the text).
  The location can be read:
  (i) from the place.name,
  if place is null or place name is null or empty,
  (ii) from the time_zone field
  if the time_zone contains the string **time** (case insensitive matching) or is null,
  (iii) then that tweet must be ignored
  [Please ensure that when you check for 'time', it is a full word.
  One approach:
  Match lowercase placename with the regex '\btime\b'
  ]
  We will ask your web service for the 'm→n' most popular hashtags.
  You should send them in descending order of popularity--primary sort key.
  The secondary sort key (in case two hashtags have the same count) is the tweet id.
  The tertiary key is the index (the position where they occur in the same tweet).

  Within a line (for one hashtag), sort tweet ids numerically (in ascending order)

  Count a unique hashtag only once per tweet (it's first occurrence).

  (m is always <=n)

  - REST format:
    GET /q4?date=yyyy-mm-dd&location=placename&m=minrank&n=maxrank
    - Example:
      http://webservice_public_dns/q4?date=2013-01-01&location=Pittsburgh&m=1&n=3

  - response format:
    ```
    TEAMID,AWS_ACCOUNT_ID1,AWS_ACCOUNT_ID2,AWS_ACCOUNT_ID3
    CMURocks:<All relevant tweet ids separated by commas>
    AwesomeSCS:<All relevant tweet ids separated by commas>
    CloudTime:<All relevant tweet ids separated by commas>
    ```

Minimum throughput requirement: 4500 rps

Sample shared at : s3://15619samples/F14/q4.sample.gz

- **HotOrNot (q5)**
  This query asks you to estimate the "hotter" user amongst two users provided.
  The Hotness Quotient is determined by the following:
  Score 1: Each tweet (#tweets): +1
  Score 2: Every time this user's tweet was retweeted (#retweets): +3
  Score 3: Every unique person who retweeted this user's tweet (#retweeters): +10
  Score: S1 + S2 + S3

  Apart from overall popularity, we also want to know the winner for each individual category. This will become clearer as you read further:

  - REST format:
    GET /q5?m=<USERIDA>&n=<USERIDB>
    - Example:
      http://webservice_public_dns/q5?m=12345678&n=987654321

  - response format:
    `TEAMID,AWS_ACCOUNT_ID1,AWS_ACCOUNT_ID2,AWS_ACCOUNT_ID3`
    `<USERIDA>\t<USERIDB>\tWINNER\n`
    `<SCOREA1>\t<SCOREB1>\t<WINNER1>\n`
    `<SCOREA2>\t<SCOREB2>\t<WINNER2>\n`
    `<SCOREA3>\t<SCOREB3>\t<WINNER3>\n`
    `<SCORE>\t<SCORE>\t<WINNER>\n`

    `If both users are tied for any of <WINNER[1,2,3]> print X`
    `Count duplicate tweets only once.`

  - Example:
    Consider the following tweet metadata from our dataset:

Tweet ID	User ID	Is Retweet?
1001	A	N
1002	B	N
1003	A	N
1004	B	N
1005	A	N
1006	B	N

Tweet ID	Retweeted by [user_id]	Original User ID [retweeted_status][user_id]	Is Retweet?
2001	C	A	Y
2002	D	B	Y
2003	A	A	Y
2004	E	B	Y
2006	F	B	Y
2007	C	A	Y
2006	F	B	Y

We come to the following conclusions:

Score Type	**Score**
A1	4
A2	9
A3	20
A -- Overall	33
B1	3
B2	9
B3	30
B -- Overall	42

Query: GET /q5?m=A&n=B
response:

```
TEAMID,AWS_ACCOUNT_ID1,AWS_ACCOUNT_ID2,AWS_ACCOUNT_ID3
A       B       WINNER
4       3       A
9       9       X
20      30      B
33      42      B
```

Minimum throughput requirement: 7500 rps

- **Shutter Count (q6)**
  This query asks you to provide the number of photos (check entities -> media --> type == photo) uploaded by users within a range of userids. {Note: Media is an array}
  - REST format:
    GET /q6?m=<USERID1>&n=<USERID2>
    - Example:
      http://webservice_public_dns/q6?m=12345678&n=987654321
  - response format:
    `TEAMID,AWS_ACCOUNT_ID1,AWS_ACCOUNT_ID2,AWS_ACCOUNT_ID3`
    `<COUNT>`

  Minimum throughput requirement: 7500 rps

Consider the following users:

User Id	Photos uploaded
1001	1
1009	1
1011	1
1012	4
1013	4
1014	8
1015	8
1016	15
1017	16
1025	23
1099	42

Expected counts are as follows:

m	n	Answer
1001	1001	1
1020	1100	65
-10	1009	2
1005	1001	0

[UPDATE1110 : Q5/Q6 helpers]

For Q5, a sample:

UID	Score1	Score2	Score3	Score
12	3	495	1630	2128
13	2	309	1030	1341
14	0	27	90	117
15	1	15	50	66
16	1	54	180	235
17	2	9	30	41
18	1	12	40	53
20	0	342	1140	1482
21	0	99	330	429
22	2	57	190	249
23	0	6	20	26
24	0	27	90	117
31	0	18	60	78
47	6	15	50	71
52	0	60	200	260
53	0	3	10	13
55	1	0	0	1
57	7	75	240	322
58	1	9	30	40
59	4	6	20	30
60	0	3	10	13
61	0	12	40	52
62	1	3	10	14
67	0	21	70	91
76	14	18	40	72

And a sample response, for m=1001395310 n=100169406 the output is:

```
1001395310   100169406    WINNER
0        21       100169406
33    12      1001395310
100  40      1001395310
133  73      1001395310
```

For Q6, a single sample should be enough:

m=0 n=9999999999

48856657

Tasks

Task 1: Front End

This task requires you to build the front end system of the web service. The front end should accept RESTful requests and send back responses.

Design constraints:

- Execution model: you can use any web framework you want.
- Spot instances are highly recommended during development period, otherwise it is very likely that you will exceed the overall budget for this step and fail the project.

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 comes close to satisfying 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: ETL

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 JSON Twitter dataset of tweets stored on **S3,** containing about 200 million tweets. The transform step applies a series of functions on the extracted data to realize the data needed in the target database. The transform 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 should backup your database to save cost. If you use EMR, you can backup HBase on S3 using the command:

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

This backup command will run a Hadoop MapReduce job and you can monitor it from both

Amazon EMR debug tool or by accessing the Jobtracker. 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 should use SPOT instances for this step otherwise it is very likely that you will exceed the budget and fail the project.

Recommendations: Always test the correctness of your system using a tiny dataset (example 200MB). If your ETL job fails or produces wrong results, you will be burning through your budget. 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. Ensure that your system produces correct query responses for all query types.

Task 3: Back end (database)

For your system to provide responses for q2-q6, you need to store the required data in a back-end database. Your front end system connects to the back end and queries it in order to get the response and send it to the requester.

You are going to use either **HBase** and **MySQL** in this phase. We will provide you with some references that will accelerate your learning process. You are expected to read and learn about these database systems on your own in order to finish this task.

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 should use spot instances for the back end system development.

We require you to build one back end, either with **HBase** or with **MySQL. This cannot match the one being graded for Phase 2**. You need to consider the design of the table structure for the database. The design of the table significantly affects the performance of a database.

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.

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-q6 queries.

Budget for Phase 1

You will have a budget **$35/team** for all the tasks in this phase. Again, make sure you tag all of your instances in this phase with key "***15619project***" and value "***phase1***". Additionally, all instances in your HBase cluster should be tagged with: **"15619backend"** and value **"hbase"**. And all instances with MySQL installed should be tagged with: **"15619backend"** and value **"mysql"**. It is up to you to decide how to spend the budget in order to get the best performance.

Budget for Phase 2

You will have a budget of **$45/team** for all the tasks in this phase. You will receive a 10% penalty if you spend between $45 and $60. 100% penalty for spending over $60.

Again, make sure you tag all of your instances in this phase with key "***15619project***" and value "***phase2***". Additionally, all instances in your HBase cluster should be tagged with: **"15619backend"** and value **"hbase"**. And all instances with MySQL installed should be tagged with: **"15619backend"** and value **"mysql"**. It is up to you to decide how to spend the budget in order to get the best performance.

Each run (test submitted to the website) must have a maximum budget of $1.25 (include on-demand EC2, storage and ELB costs. Ignore EMR and Network, Disk I/O costs). Even if you use spot pricing, the constraints that apply pertain to on-demand pricing.

Budget for Phase 3

You will have a budget of **$65/team** for all the tasks in this phase. You will receive a 10% penalty if you spend between $65 and $100. 100% penalty for spending over $100. The Live Test is not a part of this budget.

Again, make sure you tag all of your instances in this phase with key "***15619project***" and value "***phase3***". Additionally, all instances in your HBase cluster should be tagged with: **"15619backend"** and value **"hbase"**. And all instances with MySQL installed should be tagged with: **"15619backend"** and value **"mysql"**. It is up to you to decide how to spend the budget in order to get the best performance.

Each run (test submitted to the website) must have a maximum budget of **$1.6** (include on-demand EC2, storage and ELB costs. Ignore EMR and Network, Disk I/O costs). Even if you use spot pricing, the constraints that apply pertain to on-demand pricing.

Recommendations:

1. Use smaller instances when you do functionality 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 ETL too many times.
3. Spend some time to develop a plan on how to allocate the budget so that you can complete all tasks and get good performance.
4. **This is an opportunity for you to practice working as a team. For example, assign clear tasks to team members, communicate regularly, update each other on progress, specify clear interfaces, etc… All members should pull their own weight.**
5. Have fun! This should be a challenging but a rewarding learning experience.

Deliverables

Phase 1

You need to make sure you have **at least 1 test record on the testing system for q1 and 2 for q2 respectively which meet the minimum throughput requirements**. For q2, you need to have at least two submissions, one for MySQL and another for HBase. You just need to make both of them pass the Target Throughput. The submission time of the record on the testing system must be before the deadline of this phase. Besides, please submit

1. Phase 1 checkpoint report in PDF format (Phase1 checkpoint report template)
2. All your code written in this phase (front end, ETL, etc...)

To submit your work:

1. Package your completed Phase 1 checkpoint 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 the testing system.
3. This is due at midnight, Friday , October 24, 2014

Phase 2

Phase 2 grading is based on (a) Live Test Results and (b) Report

(a) Live Test

1. We will run a test for 6 hours with all teams evaluated simultaneously.
2. You need to submit your front-end's DNS address at these URLs:
   https://15619project.org/submissions/1/4/
   https://15619project.org/submissions/1/5/

To submit your work:

1. Prepare your system by 11 PM on Thursday, November 6, 2014
2. Submit your front-end URL by 11 PM on Thursday, November 6, 2014
3. The Live Test will begin an hour later (12.00.01 AM on Friday, November 7, 2014)

(b) Report

1. Phase 2 checkpoint report in PDF format (Phase2 checkpoint report template)
2. The report should explain your Live Test performance (and how it compares with your pre-LiveTest performance)
3. All your code written in this phase (front end, ETL, etc...)
4. This is due at midnight, Friday , November 7, 2014

To submit your work:

1. Package your completed Phase 2 checkpoint report and all the source code files (and a README) you wrote in the phase into a single TEAM_NAME.zip file.
2. Upload the TEAM_NAME.zip file to the testing system.

Phase 3

Phase 3 grading is based on (a) Live Test Results and (b) Report

(a) Live Test

1. We will run a test for 4 hours with all teams evaluated simultaneously.
2. You need to submit your front-end's DNS address at these URLs:
   https://15619project.org/submissions/1/8/
   https://15619project.org/submissions/1/9/

To submit your work:

1. Prepare your system by 11 PM on Thursday, November 20, 2014
2. Submit your front-end URL by 11 PM on Thursday, November 20, 2014
3. The Live Test will begin an hour later (12.00.01 AM on Friday, November 21, 2014)

(b) Report

1. Phase 3 checkpoint report in PDF format (Template)
2. The report should explain your Live Test performance (and how it compares with your pre-LiveTest performance). Make sure that you highlight how you optimized the database of your choice to maximize your throughput for each query. Your report should demonstrate a solid understanding of what you have accomplished and justifications for making certain decisions. It's a technical report which should include design choices and justifications, optimization decisions and justifications, evaluations, etc. The better the report the more confidence we will have in your understanding of your accomplishments.
3. All your code written in this phase (front end, ETL, etc...)
4. This is due at 11.59 PM, Friday , November 21, 2014

To submit your work:

1. Package your completed Phase 3 checkpoint report and all the source code files (and a README) you wrote in the phase into a single TEAM_NAME.zip file.
2. Upload the TEAM_NAME.zip file to the testing system.

Grading

Phase 1 accounts for 10% of the total grade for this 15619Project. You need to finish all the tasks in order to move on to the next phase which will add new queries. Your success on the next phases is highly dependent on how much you achieve in Phase 1. So, even though it only accounts for 10%, Phase 1 has a huge impact on the overall success of the project and should be taken very seriously.

Phase 2 accounts for 30% of the total grade for this 15619Project. You need to finish all the tasks in order to move on to the next phase which will add new queries. **You can choose any one of the backends for your final grade. If you choose to be graded on the performance of HBase, you must use only MySQL for Phase 3. If you choose to be graded on the performance of MySQL, you must use only HBase for Phase 3. You can mention which backend you want to be graded upon in your report.**

Phase 3 accounts for 60% (45% Live Test and 15% report) of the total grade for this 15619Project. **You cannot use the database that you selected for Phase 2.**

Hints and References

Front End Suggestions:

Although we do not have any constraints on the front end, performance of different web frameworks vary a lot. Choosing a slow 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 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 interactions with the back end. Please also think about whether the front end framework you choose has API support for MySQL and HBase.

General Guidelines 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 massive dataset and the poor design of your ETL process. Due to many reasons that lead to 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 a single process/thread is a waste of time and computing resources, MapReduce may be your friend, though other methods work so long as they 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