Phase 1

Writeup (/writeup/5/1/) | Submissions (/submissions/5/1/) | ScoreBoard (/scoreboard/5/1/)

Phase	Open	Deadline
Phase 1	Feb. 26, 2015 00:01	Mar. 18, 2015 23:59

Twitter Analytics on the Cloud

Learning 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.

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 analyzing Twitter data. The client has over a terabyte of raw tweets for your consumption. Your responsibility is to design, develop and deploy a web service that meets the throughput, budget and query requirements of the client. You do not have to use backend for Query 1.

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

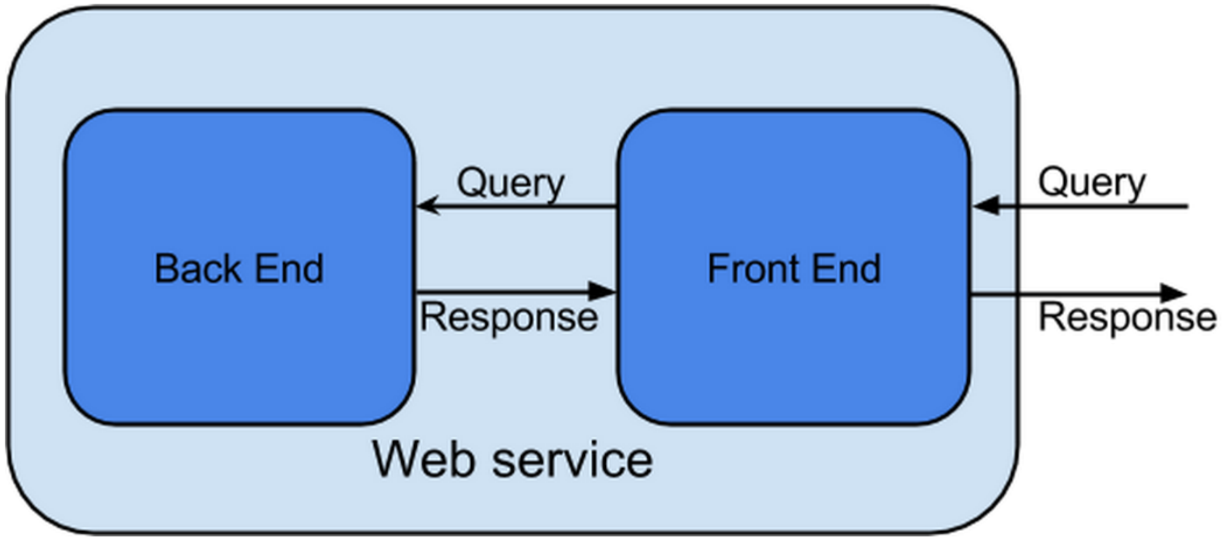

Figure 1: Block Diagram

1. A Front End: This should be a web service able to receive and respond to queries. Specifically, the service should handle incoming HTTP GET requests and provide suitable responses (as defined in the Query Types section below).
    i. 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.
    ii. 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.
    iii. The web service should run smoothly for the entire test period, which lasts for several hours.
    iv. 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.

    i. You will evaluate SQL (MySQL) and NoSQL (HBase) databases in the first two phases of this project.
    ii. 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** or **m3.large** instances for your web service (both your front-end and back-end systems). You should use spot instances for batch jobs and development.

Dataset

The dataset is at **s3://15619s15twittertest/**

(Update: s3://15619S15/raw-tweets/ has capital letters in bucket name, therefore using this s3 folder may cause unexpected errors. If you have completed ETL, do not worry, the contents in both folders are exactly the same)

It is larger than 1 TB, though we have compressed it. There may be **duplicate** or **malformed** records that you need to account for.

The dataset is in JSON (http://en.wikipedia.org/wiki/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. Twitter API (https://dev.twitter.com/docs/platform-objects/tweets)

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

Timeline & Deliverables

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.

The time allotted for the phases is as follows:

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

Load Generation and Web Service Testing

Scoreboard

We provide a real-time scoreboard (/scoreboard/5/1/) for you to compare your performance with 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 web service 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.

FAQs (we will update these to reflect common administrative questions)

- 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.
- Can I schedule future tests or multiple tests?
  - No. We want to make sure our tests are used efficiently.

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, 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.

Interpreting your result

There are several parameters shown for each query submitted:

1. Throughput: average queries per seconds during the testing period
2. Latency: Average latency for each query issued during the testing period
3. 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.
4. 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.

Query Score Calculation

The raw score for a query is based on the effective throughput for that query

Effective Throughput = Throughput * (100 - Error / 100) * (Correctness / 100)

Raw Score = Effective Throughput/Target Throughput

The Target Throughput will be provided by us for each query. As you can see, error and correctness can severely impact your effective throughput.

Phase Score Calculation

The score is a weighted sum of your **best** score for each query in that phase

Live Tests

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.

Bug Report

If you encounter any bugs in this system or have any suggestions for improvement, feel free to post privately on Piazza.

Tasks

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 **except Vert.x**
- 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.

Recommendations

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.

Hints

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 (https://www.techempower.com/benchmarks/) 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.

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:

```
 aws emr create-hbase-backup --cluster-id j-3AEXXXXXX16F2 --dir s3://mybucket/backups/j-3AEXXXXXX16F2 --
consistent
```

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 (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.

Hints

Think about your schema design before attempting 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.

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.

Back end

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 both **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. 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 loading your entire dataset. The functionality test ensures that your database can correctly produce the response to the q2-q6 queries.

References

MySQL

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

HBase

- https://hbase.apache.org/ (https://hbase.apache.org/)
- OLI Unit 4
- Project 3 and Project 4

Phase 1

Resource Tagging And Budgets

Tag all of your instances with `Key: 15619project` and `Value: phase1` for all resources

In addition to the tag above, all instances in your HBase cluster should be tagged with `Key: 15619backend` and `Value: hbase` , and all instances with MySQL installed should be tagged with `Key: 15619backend` and `Value: mysql` .

You can use any instances for ETL and debugging.

You can use only **m1.large** or cheaper instances (based on on-demand pricing) for your web service.

You can use any free AMI as your base. You should be building your own AMIs for this project.

Each run (test submitted to the website) must have a maximum budget of **$1.25/hour** (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.

You will have a budget **$40/team** for all the tasks in this phase.

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.

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]

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 (http://goo.gl/mbxyJE).

Query 1(Heartbeat and Authentication)
Target Throughput: 15000 rps
Deadline: 11:59 PM EST, Wed March 4

The query asks about the state of the web service. The front end server responds with the project team name, AWS account ids(12 digit) 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. It also authenticates the server, as a client can send it messages, which are only accessible given the correct secret key.

Your team must use the secret key:

```
X =
8271997208960872478735181815578166723519929177896558845922250595511921395049126920528021164569045773
```

For each request to q1, the load generator will generate a large message key `Y`, which is used to encrypt the message. To make sure that you can receive this key, the LG sends you the product `XY`. Your web service need to find `Y = XY / X`

Now that you have the key `Y` used to encrypt the message, you may wonder what protocol to use. AES? DES? No!!! Despite warnings from the SIGINT community, we realize that proprietary encryption is the only secure way to hide from the MSB (since they've cracked everything else!!!) Hence we use the mythical Phaistos Disc Cipher(PDC)

PDC is designed for uppercase English (A-Z) messages that have a perfect square length (4,9,16,25...). We will use a toy example to demonstrate how the encrpytion scheme works. Remember, you will must implement decryption-- given the key and the ciphertext, you must retrieve the message.

PDC has two steps: Caesarify and Spiralize.

1. In the Caesarify step, a minikey `Z = 1 + Y % 25` (where % is the modulo operator) is derived from the message key `Y`. Using `Z`, the characters of the message `M` are linearly shifted (see Figure 2) to produce the intermediate text `I`.
2. In the Spiralize step, the message is now written into a square array and read back. This reorders the characters of your message (see Figure 2). The final message produced is the ciphertext `C`.

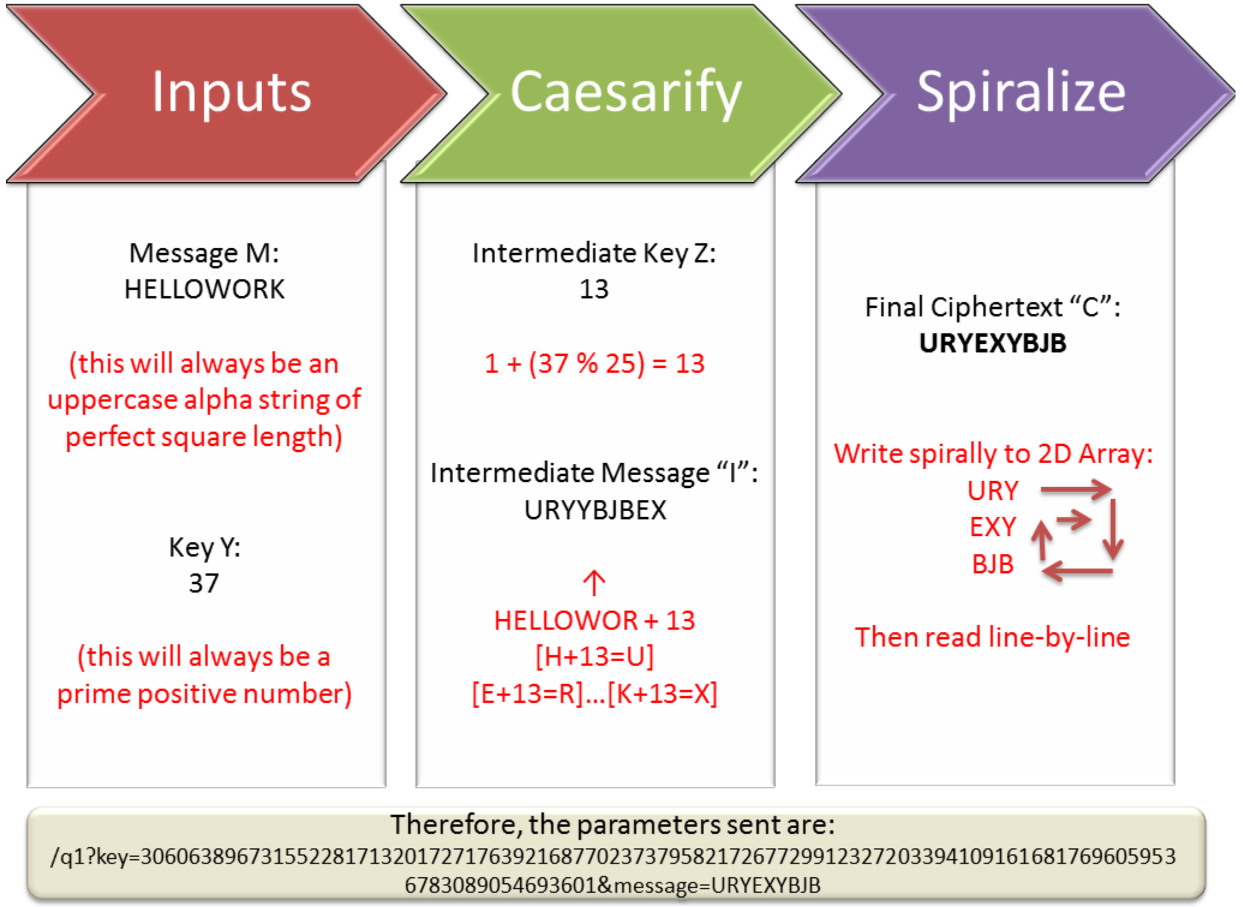
Inputs
Caesarify
Spiralize
Message M:
HELLOWORK
(this will always be an uppercase alpha string of perfect square length)
Key Y:
37
(this will always be a prime positive number)
Intermediate Key Z:
13
1 + (37 % 25) = 13
Intermediate Message "I":
URYYBJBEX
HELLOWOR + 13
[H+13=U]
[E+13=R]…[K+13=X]
Final Ciphertext "C":
URYEXYBJB
Write spirally to 2D Array:
URY
EXY
BJB
Then read line-by-line
Therefore, the parameters sent are:
/q1?key=306063896731552281713201727176392168770237379582172677299123272033941091616817696059536783089054693601&message=URYEXYBJB

Figure 2: Phaistos Disc Cipher

Remember, your web service must implement the reverse process: Given the ciphertext `C` and the key `XY` , you must derive the minikey `Z` and the message `M`

Request Format

```
GET /q1?key=<large_number XY>&message=<uppercase_ciphertext_message_C>
```

Response Format

```
TEAMID,AWS_ACCOUNT_ID1,AWS_ACCOUNT_ID2, AWS_ACCOUNT_ID3\n
yyyy-MM-dd HH:mm:ss\n
<The decrypted message M>\n
```

The time in response should be the current time in Pittsburgh

Sample Request

```
GET /q1?key=3060638967315522817132017271763921687702373795821726772991232720339410916168176960595
36783089054693601&message=URYEXYBJB
```

Sample Response

```
TeamCoolCloud,1234-0000-0001,1234-0000-0002,1234-0000-0003
2004-08-15 16:23:42
HELLOWORK
```

You must satisfy the minimum effective throughput requirement of 15000 rps for full credit

Query 2(Text Cleaning and Analysis)
Target Throughput: 5500 rps
Deadline: 11:59 PM EST, Wed March 18

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).

- The tweet id
- The sentiment score of that tweet
- The censored text

Each of these are explained in the following paragraphs.

1. The tweet id can be fetched from the `id` or the `id_str` field.

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** (http://en.wikipedia.org/wiki/Alphanumeric) character(0-9,A-Z,a-z) is encountered.

Time Filtering

Ignore all tweets that have a time stamp(`created_at` field) prior to **Sun Apr 20 00:00:00 +0000 2014**

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. AFINN (http://www2.imm.dtu.dk/pubdb/views/publication_details.php?id=6010) 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. Download it here (https://s3.amazonaws.com/15619s15files/afinn.txt). Exclude phrases in the file.

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 (http://en.wikipedia.org/wiki/ROT13) 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.

The censored list is filtered from last year's students' colorful posts on Piazza near the end of the 15619 (all these students started working on their project late). Download it here (https://s3.amazonaws.com/15619s15files/banned.txt).

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 example, 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).

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).

Request format:

```
GET /q2?userid=uid&tweet_time=timestamp
```

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.

Sample Request

```
GET /q2?userid=100001002&tweet_time=2014-05-07+02:45:56
```

Sample Response

```
Team,1234-5678-1234,1234-5678-1234,1234-5678-1234
463872330825867265:0:When you own the world
You're always home.
```

Query Reference Server

To help your team through the 15619Project design, implementation and system test, we provide a query reference server (http://ec2-54-85-149-134.compute-1.amazonaws.com)

Using the query reference server, you can submit queries Q1, Q2 to study the expected results. Access to this server is restricted to teams who demonstrate progress to the course staff in a weekly meeting with an assigned TA who will be your team's mentor.

Your team should meet the assigned TA mentor every week to talk about your progress and challenges, as well as the contribution of each member of the team. A TA will email you to set up a meeting time (ideally, during the TA office hours). Team members should also expect spot meetings with the course staff to make sure that all team members are contributing to the project.

After the weekly meeting with the TA mentor, your team will be handed an authentication token with a limited time validity, which you can use to submit requests against the query reference server.

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.

Value	Target	Weight(Grading)	Weight(Scoreboard)
Q1	15000	15%	20%
Q2(MySQL)	5500	30%	40%

Q2(HBase)	5500	30%	40%
Report	Excellence	25%	-

Grading Penalties

The following table outlines the violations of the project rules and their corresponding grade penalties for Phase 1

Violation	Penalty of the project grade
Using more than $40 to complete this phase	-10%
Using more than $60 to complete this phase	-100%
Publishing your code to public(e.g. Public Repository on Github)	-100%
Copying code from Internet without reference, from other teams or from solutions from previous semesters	-100%
Any kind of collaboration across teams	-100%

Additional Resources and References

Resources

1. Benchmarks of web servers (https://www.techempower.com/benchmarks/)
2. Schwartz, B., and P. Zaitsev. "A brief introduction to goal-driven performance optimization." White paper, Percona (2010). (http://www.percona.com/blog/2010/05/04/goal-driven-performance-optimization-white-paper-available/)
3. Practical MySQL Performance Optimization (http://www.percona.com/resources/mysql-webinars/practical-mysql-performance-optimization)
4. HBase Cheat Sheet (http://refcardz.dzone.com/refcardz/hbase)

Additional References

These are interesting papers that deal with the theory and the core problems that you will be solving for the 15619 project. You may choose to read them if you want to understand more about how Internet-scale companies (Google, Facebook, Twitter) achieve performance at scale.

Architecting web servers

1. Erb, Benjamin. "Concurrent programming for scalable web architectures." Informatiktage. 2012. (http://vts.uni-ulm.de/docs/2012/8082/vts_8082_11772.pdf)
2. Pariag, David, et al. "Comparing the performance of web server architectures." ACM SIGOPS Operating Systems Review. Vol. 41. No. 3. ACM, 2007. (https://www.ece.cmu.edu/~ece845/docs/pariag-2007.pdf)
3. McGranaghan, Mark. "Threaded vs Evented Servers" (http://mmcgrana.github.io/2010/07/threaded-vs-evented-servers.html)
4. Hu, James C., Irfan Pyarali, and Douglas C. Schmidt. "Measuring the impact of event dispatching and concurrency models on web server performance over high-speed networks." Global Telecommunications Conference, 1997. GLOBECOM'97., IEEE. Vol. 3. IEEE, 1997. (https://www.dre.vanderbilt.edu/~schmidt/PDF/globalinternet.pdf)

Clustering web servers

1. Schroeder, Trevor, Steve Goddard, and Byrov Ramamurthy. "Scalable web server clustering technologies." Network, IEEE 14.3 (2000): 38-45. (http://digitalcommons.unl.edu/cgi/viewcontent.cgi?article=1083&context=csearticles)
2. Cardellini, Valeria, Michele Colajanni, and S. Yu Philip. "Dynamic load balancing on web-server systems." IEEE Internet computing 3.3 (1999): 28-39. (http://www.ics.uci.edu/~cs230/reading/DLB.pdf)
3. Paudyal, Umesh. "Scalable web application using node.js and couchdb." (2011). (http://uu.diva-

portal.org/smash/get/diva2:443102/FULLTEXT01.pdf)

Optimizing a Multi-tier System

1. Fitzpatrick, Brad. "Distributed caching with memcached." Linux journal 2004.124 (2004): 5. (http://www.linuxjournal.com/article/7451)
2. Graziano, Pablo. "Speed up your web site with Varnish." Linux Journal 2013.227 (2013): 4. (http://www.linuxjournal.com/content/speed-your-web-site-varnish)
3. Reese, Will. "Nginx: the high-performance web server and reverse proxy." Linux Journal 2008.173 (2008): 2. (http://www.linuxjournal.com/magazine/nginx-high-performance-web-server-and-reverse-proxy)

Scalable and Performant Data Stores

1. DeCandia, Giuseppe, et al. "Dynamo: amazon's highly available key-value store." ACM SIGOPS Operating Systems Review. Vol. 41. No. 6. ACM, 2007. (http://www.allthingsdistributed.com/files/amazon-dynamo-sosp2007.pdf)
2. Cattell, Rick. "Scalable SQL and NoSQL data stores." ACM SIGMOD Record 39.4 (2011): 12-27. (http://www.cattell.net/datastores/Datastores.pdf)

Web Server Performance Measurement

1. Slothouber, Louis P. "A model of web server performance." Proceedings of the 5th International World wide web Conference. 1996. (http://www.oocities.org/webserverperformance/webmodel.pdf)
2. Banga, Gaurav, and Peter Druschel. "Measuring the Capacity of a Web Server." USENIX Symposium on Internet Technologies and Systems. 1997. (http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.61.3268&rep=rep1&type=pdf)
3. Nottingham, Mark. "On HTTP Load Testing" (https://www.mnot.net/blog/2011/05/18/http_benchmark_rules)