Phase 1 Phase 1: High Performance Web Service for Data Retrieval

6 days 12 hours

Show Submission Password

- ✔ Introduction
- ✔ Overview of tasks
- ✔ Phase 1: Overview
- ✔ Phase 1: Query 1
- ✔ Phase 1: Query 2
- ✔ Phase 1: Query 3
- ✔ More Hints

Introduction

Twitter Analytics on the Cloud

Information

Learning Objectives

This project will encompass the following learning objectives:

1. Build a performant and reliable web service on the cloud within a specified budget by combining the skills developed in this course.
2. Design, develop, deploy and optimize functional web-servers that can handle a high load (~ tens of thousands of requests per second).
3. Implement Extract, Transform and Load (ETL) on a large data set (~ 1 TB) and load the data into MySQL and HBase systems.
4. Design schema as well as configure and optimize MySQL and HBase databases to deal with scale and improve throughput.

5. Explore methods to identify the potential bottlenecks in a cloud-based web service and methods to improve system performance.

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 one 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 manage to convince your team to join this competition to demonstrate your superior cloud skills to your potential client by winning the competition.

In this competition, your team needs to build (and optimize) a web service with two components, a front end serving web requests and a back end data storage system serving different queries against the data as shown in Figure 1.

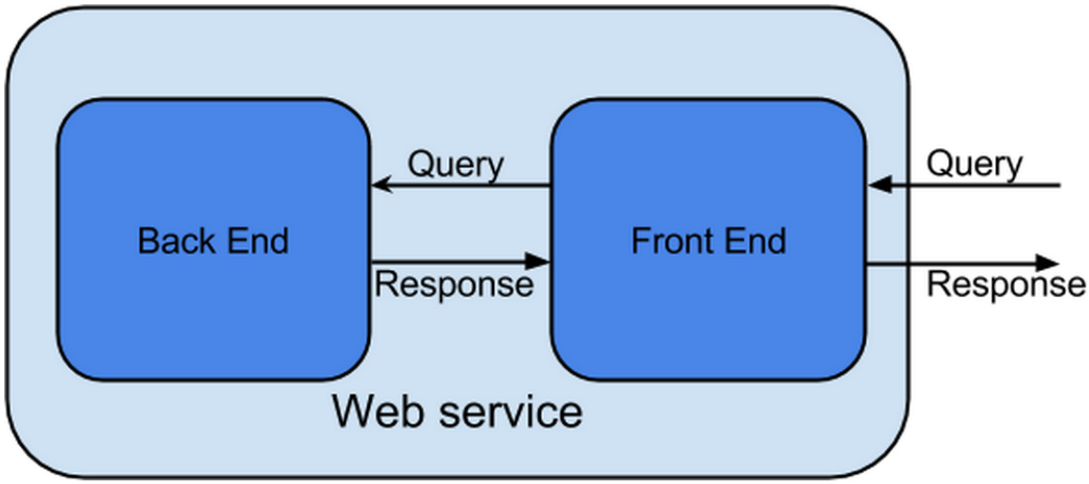

Figure 1: Architecture

Your web service design should follow these guidelines:

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).
    1. 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 the Query Types section. Query parameters are included within the URL string of the HTTP request.
    2. 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.
    3. The web service should run smoothly for the entire test period, which lasts for several hours.
    4. 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.
    1. You will evaluate both SQL (MySQL) **and** NoSQL (HBase) databases in the first two phases of this project.
    2. 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 in phase 3.
3. Your web service should meet the requirements for throughput and cost/hour 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 instances in the **M family** which are smaller than or equal to **large** for your web service (both your front-end and back-end systems). You should use spot instances for batch jobs and development. Besides, you are only allowed to use General Purpose SSD Volumes(gp2) for instance storage. Using other types of volumes(io1, st1, etc.) is not allowed.

Dataset

The dataset is at **s3://cmucc-datasets/twitter/s17/**(AWS), **https://cmuccpublicdatasets.blob.core.windows.net/twitter/s17/** or **wasb://twitter@cmuccpublicdatasets.blob.core.windows.net/s17/**(Azure), **gs://cmuccpublicdatasets/twitter/s17/**(GCP). You can do your ETL process on any of these platforms, but we encourage you to not use AWS, since we have a budget limit for AWS and ETL is costly. If you choose to use Azure or GCP, you can use any of your team members' accounts. We do not have a strict budget limit for Azure and GCP, but you should be careful about your balance in these platforms.

The dataset will be larger than 1 TB after decompressing. 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. Here is a link to the 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 the **json** module in the 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(including your auto-deployment script for Query 1).
4. Answer to questions associated with the phase.

5. A report for each phase. This report must include detailed evidence-supported reasoning for your design decisions and deliverables for each phase.

Load Generation and Web Service Testing

Scoreboard

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

How to submit your request?

You can submit the public DNS of your web service to start the test. The DNS you submit is the one we will send requests to, in order to do performance testing on your service. You can submit the DNS of an EC2 instance or of an Elastic Load Balancer. We provide tests of varying durations. Please think carefully about what you need to verify for each request and choose an appropriate duration. For example, if you want to just verify whether your service is reachable and is giving correct results, you probably don't need to run a long test. On the other hand, if you want to test your service to see how it would perform under a heavy load for a long duration, you probably don't want to run a short (1 or 2-minute) test.

How is my web service tested?

When you submit a request, a testing instance (similar to load generators that you launched in previous projects) will be launched by us. This instance will run a performance test for the specified query against the web service address that you provided. We will be testing both, performance and correctness. At the end of the test, the results will be displayed in your submission history table.

Wait List

Since we have a limited amount of resources and budget, 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, keep an eye on Piazza as well)

- Can I use my individual project account to submit a request?
    - No, you can only use the team account that you entered on theproject.zone.
- 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 submissions in the future or multiple submissions?
    - No.
- Why my 10-minute submission aborts very quickly and I got 0 score for that?
    - Our auto grader will automatically drops your submission at the 90th second if your submission have one of these:
    - < 20% of the target RPS
    - < 50% correctness

- > 20% error rate

Submission History

When your submission is running, you can check the submission history page to see the progress of the current submission 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 submission:

1. Throughput: average responses 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 the HTTP header of your response must be 2xx. Otherwise it will be recognized as an erroneous response and counted in the 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.

Phase 1, this phase, accounts for 20% of the grades of this Team Project, Phase 2 accounts for 30% of the grades, and phase 3 accounts for 50% of the grades.

Phase 1 Query Score

In Phase 1, only the score of a 10-minute submission will be counted as the score for this query. We will choose your highest score as your final score by the deadline of each query, so keep trying!

Live Tests

Phases 2 and 3 culminate in a Live Test, where services developed by all competitors (teams) are tested at the same time. Your grade for these two 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.

Overview of tasks

Overview of 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, but you **must** compare at least two web frameworks and summarize them in your report for this phase.
- Spot instances are highly recommended during the development period, otherwise it is very likely that you will exceed the overall budget for this phase and fail the project.

Recommendations

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. These hints can help you 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. You may find this resource helpful, Techempower (https://www.techempower.com/benchmarks/), it provides a detailed benchmark for mainstream frameworks.

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.

In your report, you need to be able to convince us that your team has done enough exploration and detailed comparison of different frameworks which led your team to make an informed decision. Your team should include evidence of this performance comparison.

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-<wbr>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 this 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 the ETL 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 using the larger dataset, 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 the schema design of your database before attempting the ETL job. 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. Think about possible ways to reduce the time and cost of the ETL job since you might have to run it several times.

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 could lead to the failure of your ETL job, please start thinking about your database schema and your ETL job 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 as long as they can accelerate your ETL process.

Back end

For your system to provide responses for q2-q5, 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 then sends the response 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 are not allowed to use Amazon Database Services like RDS or Cassandra for this project. 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-q5 queries.

References

MySQL

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

HBase

- https://hbase.apache.org/ (https://hbase.apache.org/)
- Project 3

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 Team 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)

Phase 1: Overview

Phase 1

Resource Tagging And Budgets

1. Tag all of your instances with `Key: Project` and `Value: Phase1` for all resources.
2. In addition to the tag above, all instances in your HBase cluster should be tagged with `Key: teambackend` and `Value: hbase`, and all instances with MySQL installed should be tagged with `Key: teambackend` and `Value: mysql`.
3. In addition to the project tag above, no other tag is required for instances used to do ETL jobs.
4. You can use **ANY** instances for ETL and debugging.
5. You can ONLY use instances in the **M family** which are **smaller than or equal to large** for your web service in submission (both your front-end and back-end systems).
6. You can use any free AMI as your base. You can also build your own AMIs for this project.
7. Your web service (each test submitted to the website) must have a maximum cost of **$0.88 per hour** (this includes on-demand EC2, EBS and ELB costs. Ignore S3, EMR, Network and Disk I/O costs). **Even if you use spot pricing, the constraints that apply to your web service pertain to on-demand pricing.**
8. You will have a budget **$50/team** for all the tasks in this phase (Phase 1).

9. For each query, we will take the highest score in your 10-minutes submissions as your score for this query.

Grading

Phase 1 accounts for 20% of the total grade for this Team Project. You need to finish all the tasks in order to move on to the next phase which will add a new query. Your success in the next phases is highly dependent on how much you have explored, learned and achieved in Phase 1. So, even though it only accounts for 20%, Phase 1 has a huge impact on the overall success of the project and should be taken very seriously. Think of it as an opportunity to learn and explore at a reduced cost.

Report

You need to submit a report at the end of phase 1, which is **23:59 ET on Apr. 4th**. This accounts for 25% of your score in phase 1. The report must conform to the template.

You can get the report template from here (https://docs.google.com/document/d/1N2U1UM3Mn5FcOGQ0oDaJvp2BT-IzyrKb5x-12O-jSvI/edit?usp=sharing).

Please work on the report as you work on the queries. The report template has some hints that guide your development. Also, you need to collect data in order to finish the report. The best way to do the project is to continuously work on your report. It will guide your progress. Historically, teams that write the report at the end of each module tend to perform badly.

Warning

In the team project, you have larger budgets but also more tasks. You should always plan your tasks, calculate the potential cost before using any resource and pay careful attention to your budget. Keep in mind that you are allowed to use AWS, Azure or GCP for the ETL process.

Warning

You are NOT ALLOWED to use ANY existing caching applications (Redis, Memcached, etc.), third-party cache libraries or ANY existing databases except MySQL and HBase in the team project. (Amazon RDS is not allowed to use.) However, you are allowed to write your own cache application manually. ANY violation of this rule will result in penalties (-100% at least).

Information

Hints:

ETL: You can use any instance types in the ETL phase, be mindful of your expenditure and budget. If we take AWS as an example, other instance families in EMR, such as the c family (Compute Optimized) and r family (Memory Optimized), may be appropriate based on your algorithm. Don't restrict your choice only to the m family (General Purpose) machines for the ETL process.

MySQL: you can refer to this (https://dev.mysql.com/doc/refman/5.7/en/optimization.html) for MySQL 5.7 official optimization document. You may switch to other MySQL versions if needed.

HBase: Please review the writeup of NoSQL primer and P3.1 and recall what you have done in P3.1. These experiences will be very useful when you design your HBase schema. You can choose to set up an HBase cluster either using EMR or manually using EC2. This has to be done on AWS. Please consider the potential performance benefit of deploying your own HBase cluster as well as the cost before making any decisions. Refer to this (http://hbase.apache.org/book.html#standalone_dist) if you want to set up an HBase cluster manually. You may also need to install Hadoop and ZooKeeper before installing HBase. You can also try other Hadoop distributions such as CDH (https://www.cloudera.com/downloads/cdh/5-10-0.html) developed by Cloudera.

Information

Reference Server:

We have also provided you a reference server (http://reference.15619project.org/) for you to check the correctness of your results (don't forget to put in query URL). We highly recommend you to use this server for checking the correctness of your output files before loading to your database.

You can also use this server to figure out possible encoding problems that you may encounter in Q2.

Information

Value	Target RPS	Weight
Q1	28000	15%
Q2 (MySQL)	7000	15%
Q2 (HBase)	7000	15%
Q3 (MySQL)	1500	15%
Q3 (HBase)	1500	15%
Report	-	25%

Danger

Violation	Penalty of the project grade
Using more than $50 to complete this phase	-10%

Violation	Penalty of the project grade
Failing to tag all your resources for this project	-10%
Using more than $0.88 per hour for submissions	-2*n%(where n is the percentage of exceeded budget. e.g. spending $1.00 per hour will result in 24% penalty)
Using more than $75 to complete this phase	-100%
Submitting **ANY** kind of credentials in code	-100%
Using instances not in the M family or larger than large for your web service in submission (both your front-end and back-end systems).	-100% at least
Publishing your code to public(e.g. Public Repository on Github)	-200% at least
Copying code from Internet, other teams or solutions from previous semesters	-200% at least
Any kind of collaboration across teams	-200% at least

Phase 1: Query 1

Query 1 (Heartbeat and Authentication)

Target Throughput: **28000 RPS**

Due `11:59:59 PM ET, Sun Mar. 12`

Danger

IMPORTANT NOTE: Please **START EARLY**! Query 1 is due at `11:59:59 PM ET, Sun Mar. 12` , not `Sun Apr. 2` ! You should make at least one 600-second submission on TPZ which reaches the target throughput to get full marks for this query.

1. We require you to carry out a performance comparison of different web frameworks in Query 1 in order to let you develop a deeper understanding of the pros and cons of web frameworks.
2. You should have submission record with your implementation on at least two frameworks.
3. You will write a bash script to auto-deploy your two front-end servers and submit it with your code. The script should automatically install packages needed to run your server on one of these clean state AMIs: Ubuntu Server 16.04 LTS (ami-f4cc1de2) or Amazon Linux AMI 2016.09.1 (ami-0b33d91d), and do all configurations needed before launching your server. Please note down the distribution you choose in the

comment of the script. You can assume that we will run your script along with your code, and since we need you to submit your code, you should compile your submitted code in your script. Don't directly submit executable and run it in your script!
4. You need to compare different front-end configurations (instance types & number).
5. We encourage you to try writing your own load balancer and compare its performance with ELB.
6. **Read the report template carefully** before getting your hands dirty in the project so that you can have a better understanding of what you need to do and deliver in this phase.

Query 1 asks about the state of the web service. The front end server should respond with the project team id, AWS account id, the current timestamp and a decrypted message. It is generally used as a heartbeat mechanism, but it could be used here to test whether your front end system can handle varying loads. It also authenticates the server, as a client can send it encrypted messages, which are only accessible with the correct secret key.

Your team will use the following secret key:

`X =`
`1238908405918409830812309857928320488095680090929383122313479825749637212487923741219391823918392814 0`

For each request to *Query 1*, the load generator will generate a message key `Y`, and cipher `C`. The cipher is encrypted by key `Z`, and `Z` is derived from `X` and `Y` (elaborated below).

`Z` is generated by iteratively digit-adding part of `X` and `Y`.

Digit-add (+) applies on two numbers of the same length. It works by adding each digit in one number to the corresponding digit in another and then modulo 10. For example, 4859 + 1303 = 5152, as shown below in Figure 2.

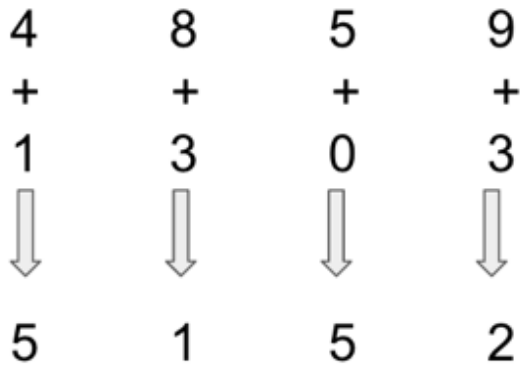

Figure 2: Digit-add example

Now let's see how we can get `Z` from `X` and `Y`. Suppose `X` is n-digit long and `Y` is m-digit long (it is guaranteed that n is not smaller than m). To get `Z`, we iteratively digit-add `Y` with every m-digit long number starting at the first n-m+1 positions of `X`. See the following example. At each step, we digit-add `Y` with part of `X`, then move `Y` one digit to the right. We repeat this until we reach the end of `X`.

Suppose `X` =134859037, `Y` =6347

We first calculate 6347+1348=7685, because 1348 is the first 4 digits in `X`. This gives 7685. Then we calculate 7685+3485=0060, because 3485 is the second 4 digits in `X`. We continue this process, 0060+4859=4819 In the last step, we have 7202+9037=6239, which is the derived key `Z`. This process is shown below in Figure 3:

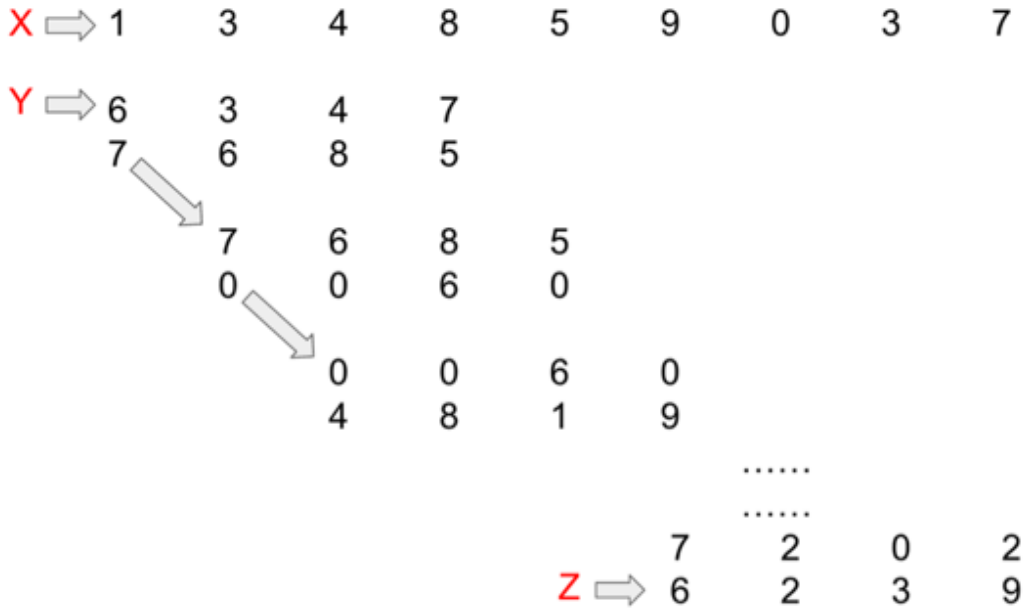

Figure 3: Key calculation example

In Query 1, we use `X` as stated before.

Now that you have the key `Z` 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 the MSB has cracked everything else!!!) Hence we use the mythical Phaistos Disc Cipher (PDC).

PDC is designed for uppercase English (A-Z) messages. We will use a toy example to demonstrate how the encryption scheme works. Remember, you will implement decryption - given the key and the ciphertext, you must retrieve the original message.

PDC has three steps: KeyGen, Caesarify and Spiralize.

In the KeyGen step, a big integer `Y` is randomly selected, we generate another encrypted key and call it `Z`. (The way to generate `Z` is stated before)

In the Caesarify step, a minikey `K` = 1 + `Z` % 25 is derived from key `Z`. Using `K`, the characters of the message M are circularly shifted to produce the intermediate text `I`. For example, if `K` =10, ‘A’ will be transformed to ‘K’, ‘X’ will be transformed to ‘H’. See the example below in Figure 4 and this page (https://en.wikipedia.org/wiki/Caesar_cipher).

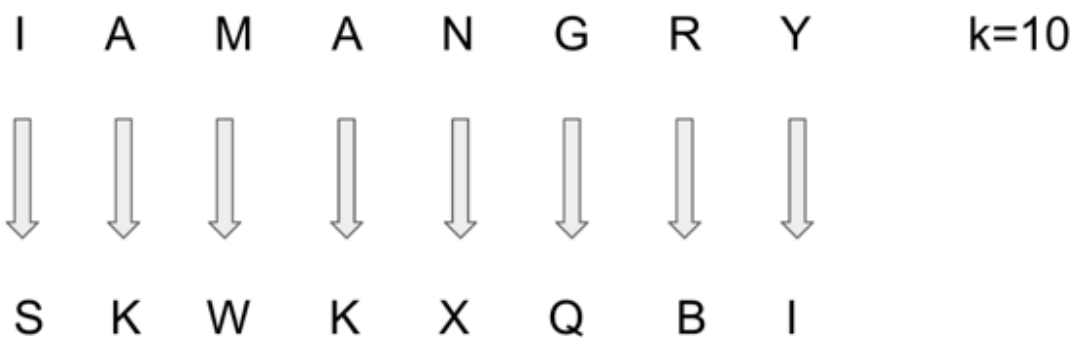

Figure 4: Caesarify example

In the Spiralize step, the intermediate text `I` is written into an equilateral triangle in level order first, then the triangle is rotated 120 degrees clockwise. This reorders the characters of this message. Then we read the triangle in a spiral way to get the final message. The final message produced is the ciphertext `C`. Examples of the equilateral triangle are shown below in Figure 5. Note that the length of the message needs to fit exactly into an equilateral triangle, or it is an invalid message.

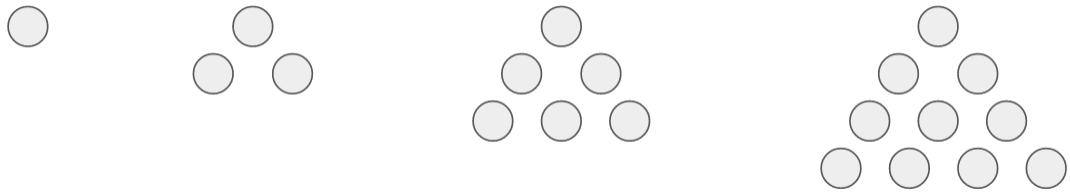

Figure 5: Examples of equilateral triangles

See the example below in Figure 6 for how the Spiralize step works on message 'ABCDEFGHIJKLMNOPQRSTU'. In this example, we read the rotated triangle in a spiral way and the final ciphertext is 'PQRSTUOJFCABDGKLMNIEH'.

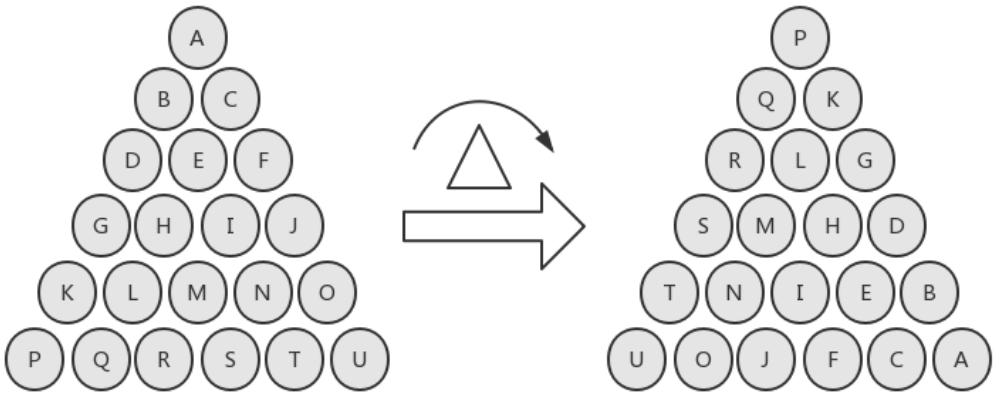

Figure 6: Spiralize example

Remember, your web service needs to implement the reverse process: Given the ciphertext `C` and the key `Y`, you must derive the key `Z` using your secret key `X`. Then use the minikey `K` to decrypt the message. Your code also needs to handle a malformed request (e.g. invalid key & message), if the request is malformed, return the decrypted message as "INVALID".

Request Format

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

Response Format

```
TEAMID,TEAM_AWS_ACCOUNT_ID\n
yyyy-MM-dd HH:mm:ss\n
[The decrypted message M]\n
```

TEAM_ID Is the team name, TEAM_AWS_ACCOUNT_ID is the aws account id of the team account

The time in the response should be the current time in Pittsburgh, Eastern Time

Sample Request

```
GET /q1?key=123979324798794871273918749230801230918402384981739718927398172391222 1
&message=QTGXGTHWEQENWQVKPIRFO
```

Sample Response

```
TeamCoolCloud,1234-0000-0001
2004-08-15 16:23:42
CLOUDCOMPUTINGFOREVER
```

Phase 1: Query 2

Query 2 (Keyword Frequency of Tweets with the Same Hashtag)

Target Throughput: **7000 RPS**

Due `11:59 PM EDT, Sun Apr. 2`

Danger

In the team project, you have larger budgets but also more tasks. You should always plan your tasks, calculate the potential cost before using any resource and pay careful attention to your budget. Allocate your budget for development and testing. Keep a buffer for potential issues that your team might encounter.

Warning

Again, you are NOT ALLOWED to use ANY existing caching applications (Redis, Memcached, etc.), third-party cache libraries or ANY existing databases except MySQL and HBase in the team project. (Amazon RDS is not allowed to use.) However, you are allowed to write your own cache application manually.

ANY violation of this rule will result in penalties (-100% at least).

Information

Hints:

ETL: You are allowed to use AWS, Azure or GCP for the ETL process. Furthermore, you can use any instance types in the ETL phase, be mindful of your expenditure and budget. For example, on AWS, other instance families in EMR, such as the c family (Compute Optimized) and r family (Memory Optimized), may be appropriate based on your algorithm. Don't restrict your choice only to the m family (General Purpose) machines.

For all queries (Q2 onwards) you will utilize a **real** Twitter data set that we collected. You can download either a partial sample or the whole data set from S3, the URL is `s3://cmucc-datasets/twitter/s17/part-r-00XXX.gz` , where XXX is from 000 to 999. The dataset is also available on Azure and GCP at the locations listed above.

In this query, we are going to explore the top N users that have posted tweets on a certain trend (represented by hashtag), together with their tweets' keyword frequency (which will be defined later). This query tests your ability to apply what you learned in the course to process a large amount of data and to build and integrate an efficient back end database with your front end web server.

The request of Q2 provides one hashtag, a number N and a list of keywords, for which you need to respond with:

The top N users(the ranking defined below) whose tweets contain the given hashtag and the frequency of the provided keywords in those tweets (defined below).
NOTE:if there are fewer than n users satisfying the requirement, just return them all

In the response, you will need to return:

- The user id
- The frequency of keywords in the user's tweets

NOTE: For the keyword frequency calculation, we only focus on the user's tweets that contain the provided hashtag.

Here is how you can obtain the information that might be useful:

- The tweet id can be obtained from either the `id` or the `id_str` fields
- The user id can be obtained from either the `id` or the `id_str` in the `user` object of each tweet record
- The tweet text can be obtained from the `text` field
- The hashtags can be obtained from the `text` field inside `hashtags` field, which is under the `entities` field

- The frequency of keywords of the user's tweets needs to be calculated using the text of each tweet (described later)

You can also learn more about the fields and their definitions from the (Twitter API) [https://dev.twitter.com/overview/api/tweets].

Duplicate Tweets

There may be duplicate tweets (tweets with the same id) in the raw data. Retain only one of them.

Hint: tweet id might be a good candidate for detecting duplicated tweets.

Malformed Tweets

Tweets that satisfy any one of the following conditions should be regarded as malformed tweets:

- Both `id` and `id_str` of the tweet object are missing or empty
- Both `id` and `id_str` in user object are missing or empty
- `created_at` field is missing or empty
- `text` field is missing or empty
- `lang` field is missing or empty
- `hashtag` text (stated above) is missing or empty
- Cannot be parsed as a JSON object

You should filter them out of the data set.

Language of Tweets

For Query 2, we are including tweets in various languages, here are the language tags for each of them: Arabic (`ar`), English (`en`), French (`fr`), Indonesian (`in`), Portuguese (`pt`), Spanish (`es`), Turkish (`tr`)

There is a field `lang` in each tweet object. Filter out the tweets whose 'lang' field is **not** listed above. Tweets whose 'lang' field is null should also be filtered out.

Shortened URLs

There are many shortened URLs in tweets, like `http://t.co/iZbYagMN6e` .

They would introduce too many unique meaningless words. We provide you with this regular expression to remove them before you do any analysis.

Python :

```
(https?|ftp):\/\/[^\t\r\n /$.?#][^\t\r\n ]*
```

Java :

```
(https?|ftp)://[^\\t\\r\\n /$.?#][^\\t\\r\\n ]*
```

Keyword Frequency Calculation

Be sure to eliminate short URLs **BEFORE** doing this step.

In order to calculate the keyword frequency of a user's tweets, we have to find a way to transform the user's tweets into a words list that is similar to the keywords list we provided.

In our case, a word is defined as one or more consecutive letters in **any kinds of languages** separated by the non-letter character(s). Words are **case insensitive**.

Text Split

To calculate scores, you may need to split the tweet text into a list of words. As before, a word is defined as one or more letter characters separated by the non-letter character(s). For example

> Cloud_computing_is_ _great

has 4 words, the same rules apply to the rest of the 6 languages.

Given that the way we define words in Query 2 is different from what you have seen in previous projects, we strongly recommend that you read the resources provided from here (http://www.regular-expressions.info/unicode.html) in order to get you started on finding words that consist of consecutive letters in all the languages we are using.

For python users, you might want to consider importing modules when you are running MapReduce. To save your time, we hereby provide you bootstrap code that you can add to pre-install modules before running MapReduce.

```
#!/bin/bash

set -e -x
sudo pip install -U regex
```

Simply copy and paste the code above into an .sh file and upload the file to AWS S3.

Here (http://docs.aws.amazon.com/emr/latest/DeveloperGuide/emr-plan-bootstrap.html) is a brief introduction of creating bootstrap actions to install additional Software in EMR. Here are the steps that get you to set up your own bootstrap action:

1. In the Elastic MapReduce view, click Create Cluster.
2. At the top of the page, click on Go to advanced options.
3. Configure your EMR as usual until you reach Step 3: General Cluster Settings.
4. Under the bottom of this page, click on "Bootstrap Actions".
5. Select "Custom Action" and click on "Configure and add".
6. Give your action a name and select the .sh file that you uploaded to S3.
7. Click on Add.

Now that you have already added your bootstrap action, you can change the rest of the setting as usual and the module will be installed before your streaming program begins.

NOTE: Modern English uses a space to separate words, but not all languages follow this practice, such as Chinese and Japanese where consecutive letters can actually be phrases or sentences. Although we filter the data by languages and keep 7 languages with the non-letter character(s) as the word divider, text in other languages can still exist in the data set. (e.g. There can be Chinese characters in a tweet with `lang` as `en`.) For Query 2, we bypass this hurdle and just tokenize words as consecutive letters.

Effective Word

To further narrow the words you need to use during the calculation, we need to identify effective words in a tweet.

We provide a list of stopwords (in all 7 languages) that should NOT be counted as an effective word. The list of stopwords can be downloaded here (https://s3.amazonaws.com/cmucc-public/s17/teamproject/stopwords.txt). Stopwords are **case insensitive**. That is, ‘I’ and ‘i’ are

both considered as stop words.

After applying the stop words rule, the example above

> Cloud_computing_is_ _great

will only have **3** effective words ("is" is a stop word).

Keyword Frequency Calculation

A keyword frequency in a tweet is represented as the occurrences of a keyword

For example, when provided with a hashtag

> #enjoycloudcomputing

with a keywords list that contain two words:

> cloud, computing

The keywords "cloud" and "computing" in the tweet below will have a frequency of 2 and 1 respectively:

> Manipulating resources on the cloud is so much fun! I love cloud computing! #enjoycloudcomputing

The total keyword frequency of the current tweet is therefore 2 + 1 = 3. We then get the user's keyword frequency by adding up all the keyword frequency of his/her **tweets containing the provided hashtag**.

Note: Consider all the words as **case insensitive** during the calculation. But for hashtag in the request, treat them as **case sensitive**.

Reference ETL result of a small data set

We have provided you with a reference ETL result of a small data set to help you make sure you get the correct words split which is the first and very important step. The reference file can be downloaded at here (https://s3.amazonaws.com/cmucc-public/s17/teamproject/reference). This is the ETL result for the first part of the data set (`s3://cmucc-datasets/twitter/s17/part-r-00000.gz`).

Note: To give you a taste on the word split, we have provided you with the comma separated split words in the `text` field, but the stopwords are **not** removed from it. You have to remove the stopwords in order to get the right word count for the keyword frequency calculation. the reference file does not give any suggestion for the ETL or database design (In fact it's far from the optimized design).

Request format:

```
GET /q2?hashtag=hashtag&N=a number&list_of_key_words=<a_string_of_comma_’,’_seperated_keywords>
```

Response format:

```
TEAMID,TEAM_AWS_ACCOUNT_ID\n
user_id_1:Keywords_frequency_1,user_id_2:Keywords_frequency_2,user_id_3:Keywords_f
requency_3,...,user_id_N:Keywords_frequency_N\n
```

Details are as follows:

Requests: You can assume that all requests have three non-empty parameters: hashtag, keywords and n (can be represented as an int)

`\n` should be replaced with the new line character `\n` instead of keeping it a `\` and `n`.

Sample Request:

```
GET /q2?hashtag=cloudcomputing&N=100&list_of_key_words=cloudcomputing,cloud,comput
ing,linux,aws,bigdata,business,storage,amazon,saas
```

Sample Response:

```
TeamCoolCloud,1234-0000-0001
936891528:65,715187934853406721:17,99035313:14,2214416916:13,...
```

We rank users based on the frequency of keywords in their tweets (in decreasing numerical order). Break ties by user id (in increasing numerical order).

Malformed Request

Your code should also handle the situation when the fields (hashtag, N, list_of_keywords) have missing values in a GET request, in such cases, you should simply respond with your team's basic information:

Sample Malformed Request:

```
GET /q2?hashtag=&N=100&list_of_key_words=cloudcomputing,cloud,computing,linux,aws,
bigdata,business,storage,amazon,saas
```

Sample Response

```
TeamCoolCloud,1234-0000-0001\n
```

Information

Hints and Clarifications

- If one hashtag appears in a tweet multiple times, the tweet should be weighted based on the frequency of this hashtag showing up. So word frequency in such tweets should be calculated multiple times if it contains the hashtag more than once.
- Treat all contents in the `text` field the same, including hashtag. Meaning if the text contains hashtags, these hashtags can be considered as a word and need to be included in the keyword frequency calculation.
- Only hashtags in the request should be considered as case sensitive.
- Be very careful when loading CSV/TSV files that contain Unicode characters into MySQL and HBase.

Query 3: Range Query and Topic Words Extraction

Target Throughput: 1500 RPS

Due: `11:59:59 PM EDT, Sun Apr. 2, 2017`

Introduction

In this query, you will dig further into MySQL, HBase, and your web-tier framework. Apart from consolidating your knowledge of large-dataset processing and web-service building techniques, you will also learn how to configure your database and web server to reach their peak performance.

Objective

The dataset for this query is the same as Query 2, you can find this data at the locations below:

- AWS: `s3://cmucc-datasets/twitter/s17/`
- Azure: `https://cmuccpublicdatasets.blob.core.windows.net/twitter/s17/` or `wasb://twitter@cmuccpublicdatasets.blob.core.windows.net/s17/`
- GCP: `gs://cmuccpublicdatasets/twitter/s17/`

For this query, we will make requests to your server that have a time range, uid range, the maximum number of topic words (`n1`), and the maximum number of tweets that should be returned (`n2`).

You will have to;

1. Find all the tweets posted by a user within the uid range AND within the given time range. ***All ranges are inclusive.***
2. Calculate the `topic score` , which is a modified version of TF-IDF (described later) to extract the topic words.
3. Sort and return at most `n1` topic words and return them along with at most `n2` sorted tweets, which must contain at least one of those `n1` topic words.

In this query, we will send requests to your front end with the following parameters:

- **`time_start`** : in Unix time / Epoch time (https://en.wikipedia.org/wiki/Unix_time) format, which marks the starting time search boundary
  - e.g. `time_start=1480000000`
- **`time_end`** : in Unix time / Epoch time (https://en.wikipedia.org/wiki/Unix_time) format, which marks the ending time search boundary
  - e.g. `time_end=1490000000`
- **`uid_start`** : marks the starting user search boundary
  - e.g. `uid_start=492500000`
- **`uid_end`** : marks the ending user search boundary
  - e.g. `uid_end=492600000`
- **`n1`** : the maximum number of topic words that should be included in the response
  - e.g. `n1=10`

- `n2` : the maximum number of tweets in all that should be included in the response
  - e.g. `n2=50`

Your response should contain:

- Your team name and team AWS ID.
- `n1` topic words sorted by their `topic score` s in descending order, breaking ties in ascending lexicographical order.
- If there are less than `n1` topic words, all topic words will appear in the response.
- A list of tweet text, each of which contain **ANY** of these `n1` topic words, in descending order by their impact scores, breaking ties by tweet id in descending order.
- If there are less than `n2` tweets, all tweets will appear in the response.

Note that your front end should always check the validity of the request parameters. It should never crash or get stuck on invalid inputs. In these cases, respond with your team name and AWS ID only.

For example, we might send requests where `n1` is not a valid Integer, and we might test SQL injection vulnerability in your front end - so make sure you use Prepared Statements!

Further requirements on the response:

- The tweets and topic words you calculate **MUST** be based on the tweets posted between `time_start` and `time_end` **AND** posted by the user whose userid is between `uid_start` and `uid_end` .
- You should always keep only two decimal places for the `topic score` of each word. You don't need to round up the number if there are more than two valid decimal places for the score.
  - i.e. 74.568 should be displayed as 74.56 in the output.
- Both the topic words and the tweet texts in your response should be censored (described later in detail).
- The returned tweet text should keep the short URLs (described later in detail).
- Response format should be the following:

```
[YOUR_TEAM_NAME],[YOUR_TEAM_AWS_ID]
word1:score1\tword2:score2...\twordn1:scoren1
impactscore1\ttid1\ttext1
impactscore2\ttid2\ttext2
 ...
impactscoren2\ttidn2\ttextn2
```

- Check out the sample responses below regarding the response format.

Sample requests and responses

Valid Request 1:

```
http://dns.of.instance/q3?uid_start=2317544238&uid_end=2319391795&time_start=14021
26179&time_end=1484854251&n1=10&n2=8
```

Response:

```
[YOUR_TEAM_NAME],[YOUR_TEAM_AWS_ID]
online:1869.58  love:1758.53    camgirl:1692.63 like:1669.67    just:1656.46    au
gust:1419.20  new:1325.65 amp:1270.26 u:1232.91   good:1175.60
871208  815093696718774273  RT @Taysbra: O*G I CANT BELIEVE THAT CAMILA ACTUALLY S
PENT THE HOLIDAYS WITH HER FAMILY O*G WHY WOULD SHE SHADE 5H LIKE THAT.
T…
864504  816381993973751812  @randoom15 The Survivor isn't a Town role. That would
 be like making it so executing a Jester, or Amne made the Jailor lose his executi
ons.
628780  627830371137097729  Retweet to gain just follow everyone who retweets and
 follow back who follows you

Follow me &amp;amp; @nikystylees to be next
625788  631201207420329989  RT @ManOnHisPath: There's nothing wrong with taking a
 lot of photos, but make sure you capture moments with your heart, not just your p
hone
569576  475957144076812288  SEO Tip of the Week: Prioritising &amp;amp; finding keywor
ds with Google keyword planner: SEO Tip of the Week: Google ... http://t.co/KSMnou
nEtK
561324  627571041536221184  RT @anxietysmind: i basically have two moods, either l
ets do something spontaneous and awesome, or lets just lay in bad all day and forg
et …
523782  630954657880223744  N.F.L. Roundup: Patriots Shake Up Their Quarterback Ro
ster: New England released Matt Flynn, who had been on t... http://t.co/D8WqOMh7vi
488978  820242557179494401  RT @MyPlace4U: Ukraine Govt Officials intervened in U.
S. Election to help get Hillary Elected. @JoeConchaTV @mtracey @brithume  https://
t.c…
```

Valid Request 2:

```
http://dns.of.instance/q3?uid_start=21919458&uid_end=22049006&time_start=139650383
0&time_end=1424866940&n1=7&n2=10
```

Response:

```
[YOUR_TEAM_NAME],[YOUR_TEAM_AWS_ID]
channel:2270.04 amp:1586.31 new:1166.24 just:1153.70    love:1063.31    like:1015.
71    good:937.63
26200650    461159182406672384  I just buyed the comedy album of my bestest friend
in the entire world @briangaar. https://t.co/hwDB4veaYG #RacesAsToad
6541500 453927321490452480  @Sheddenizen wonderful, thanks. Please do join in and
 share your photo with the book via @GuardianWitness (see piece). We'd love to see
it
6315224 454217030434951168  @DugsiDropout If you have an order you'd like for us t
o check on. Please DM us with your customer and order numbers.
6240793 468859937117175809  RT @TheKrewOfficial: Hey it's Dempsey! Check out this
 fun pic from our recent shoot of me &amp;amp; my lil sis Alijah! @TichinaArnold htt
p://t.co/N…
6225800 475291403179003904  "After awhile, your cheap talk don't even cause me pai
n, so let your bullets fly like rain." - Jimi Hendrix, Machine Gun
3838437 462267791463022592  Your Canadian Headline Of The Year: Mad Beaver Stops T
raffic In New Brunswick. http://t.co/hVXY2fb85U
3413488 461935724200464384  New Foo Fighters Album Details Revealed in Instagram P
hotos: http://t.co/vK2z9xFxZt
3304202 468764436992626688  To limit the spread of fraudulent drugs, Nigerian &am
p; Indian distributors are using a text messaging system http://t.co/1TEnfDJnHj
1670718 459365261775085568  @Chingyttr Sorry, just the Olives right now. We will u
pdate you if they become available.
1442000 467001780732428288  The inimitable @SNASKsthlm have just rocked up dressed
as a pope, a king and a cuddly bear. #TYPO14
```

Invalid Request 1:

```
http://dns.of.instance/q3?uid_start=492520982&uid_end=492620184&time_start=1483124
687&time_end=1484856522&n1=hello&n2=5
```

Response:

```
[YOUR_TEAM_NAME],[YOUR_TEAM_AWS_ID]
```

Invalid Request 2:

```
http://dns.of.instance/q3?uid_start=whatiseven&uid_end=happening&time_start=148312
4687&time_end=1484856522&n1=20&n2=word
```

Response:

```
[YOUR_TEAM_NAME],[YOUR_TEAM_AWS_ID]
```

Requirements

1. ***Tweet Filtering***
    - Remove duplicate tweets and malformed tweets like you did in Query 2, except that now `hashtag` field is not necessary to exist.
    - For language, only retain tweets whose value in language field is `en`.
2. ***Short URLs***

    Short URLs only need to be removed when you are calculating the impact score and extracting topic words. **DO NOT** remove them in the tweet text you return!

The rules for removing short URLs in Query 2 apply to Query 3;

- python: `(https?|ftp):\/\/[^\t\r\n /$.?#][^\t\r\n ]*`
- java: `(https?|ftp)://[^\\t\\r\\n /$.?#][^\\t\\r\\n ]*`

3. ***Text Censoring***

Since we have innocent young students doing this assignment, we do not want to expose them to some of the language that exists on Twitter. So we provide a mechanism to sanitize the text.

- Be sure to censor on the **ORIGINAL** text. Do **NOT** do short URL elimination for this step.
- We give you an ROT13ed version (link here (https://s3.amazonaws.com/cmucc-datasets/15619/s17/go_flux_yourself.txt)) 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.
- Since many people try to avoid censoring by using punctuation to concatenate words, for text censoring we define words as separated by **ANY NON-ALPHANUMERIC** characters, **INCLUDING** `'` and `-`. Note this is **DIFFERENT** from Query 2 and the word definition listed below used for impact score calculation and topic words extraction.

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.
```

As you can see, censored words are case insensitive. That is, “Cloud” should be censored because ‘cloud’ is in the list. However, 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). Note that the censoring should also apply to the topic words you return.

4. ***Word Definition***

- Note the word definition here is **DIFFERENT** from Query 2!
- For impact score calculation and topic words extraction, words are defined as one or more consecutive alphanumeric characters (A through Z, a through z, 0 through 9) with zero or more `'` or/and `-` characters. *However*, a word should **not** contain only numbers and/or `'` and/or `-` characters.
- Words are separated by all other character(s) not mentioned above.
- For example, the following tweet contains **6** words.

```
Query 3 is su-per-b! I'mmmm lovin' it!
```

5. ***Impact Score Calculation***

    - Remember to remove short URLs **BEFORE** this step!
    - Note the stop word list here is **DIFFERENT** from Query 2!

    To calculate the impact score, we need to identify effective words in a tweet.

    We provide a list of lowercase English stopwords that should NOT be counted as an effective word. The list of stopwords can be downloaded here (https://s3.amazonaws.com/cmucc-datasets/15619/s17/stopwords.txt).

    Stopwords are case insensitive. That is, 'I' and 'i' are both considered as stopwords.

    Effective Word Count(EWC) should be calculated as:

    ```
    EWC = total number of non-stopwords
    ```

    For example,

    ```
    I love Cloud Computing
    ```

    Has EWC of 3, since "I" is a stop word.

    Note: If a word appears multiple times, it will be counted multiple times.

    Let's define the meaning of our impact score.

    - Each tweet has a 'favorite_count' field and a 'retweet_count' field.
    - Each user has a 'followers_count' field.
    - If a record does not have a field, it means that field has a value of 0.

    Thus the impact score of a tweet is defined as:

    ```
    impact_score = EWC*(favorite_count+retweet_count+followers_count)
    ```

    If you get a negative impact_score (e.g. when one tweet has a negative value for `retweet_count` and the other two fields' values are 0), consider the impact score to be 0.

6. ***Topic Words Extraction***

    - Remember to remove short URLs **BEFORE** this step!
    - Text censoring should be done **AFTER** topic words have been chosen!
    - Topic words should be in lower case, so `cloud` and `Cloud` are considered as the same topic word.
    - Some words look the same after censoring, but they still need to be considered as different words when calculating the topic score.
    - For example, if `cloud` and `could` are both banned words, their censored text is the same (`c***d`). However, `cloud` and `could` should be considered as different topic words when you do the extraction, so you have to censor and list them separately if you include them in your response!
    - We use a variant of the classic TF-IDF algorithm to extract topic words for this query. You can read this short article (http://www.tfidf.com/) introducing the original algorithm.
    - In this query, we change the calculation a little bit. Here to calculate the TF-IDF for word `w` in tweet `t` , TF = term frequency of word `w` in `t` , and IDF = ln(Total number of tweets in range/ Number of tweets with `w` in it). Also, originally a TF-IDF

score only indicates the contribution of a word to a document, now we want you to calculate the aggregate TF-IDF score of a word to all documents (namely all the tweets in the given time range and uid range in the request).

- We define this score as `topic score`. To be specific, suppose the total number of tweets in the given time and uid range is `n`, the TF-IDF score of word `w` in tweet `Ti` is `x`, the impact score of `Ti` is `y`.

  Then the `topic score` of `w` is:

  ```
  sum(x * ln(y + 1)) (i from 1 to n)
  ```

- You should apply this algorithm and calculate the `topic score` for each word, and the higher the score is for this word, the more likely it is a topic word.
- Note that the stop words we provided do **NOT** count as topic words.

Reference the ETL result of a small data set

We have provided you with a reference ETL result of a small dataset to help you make sure you get correct word counts which is the first and most important step.

The reference file can be downloaded here (https://s3.amazonaws.com/cmucc-datasets/15619/s17/query3_ref.txt). This is the ETL result for the first part of the data set (`s3://cmucc-datasets/twitter/s17/part-r-00000.gz`).

Note: the reference file does not give any suggestion for the ETL or database design.

Information

Hints and Clarifications

- We will send Q3 requests to your web service with reasonable ranges. Try to log some requests and get an idea of the range of userIds and dates. That should help!
- Be very careful when loading CSV/TSV files that contain Unicode characters into MySQL and HBase.
- To echo our earlier statement, the ranges for of `uid` and `time` are **inclusive**. For example, if we request time to be within the range from 2013-10-10 13:00:00 to 2013-10-10 13:10:00, tweets that were posted exactly on 2013-10-10 13:00:00 and 2013-10-10 13:10:00 should be counted.

More Hints

We were lying. Just wanted you to solve the AssessMe!