Team Project Report - Phase 1

Note: We Strongly recommend reading through both checkpoint and final report template before starting the project

Phase 1 summary

Team name: **ThreeCobblers**
Members (names and Andrew IDs):
Leo Guo (jiongtig)
Benny Jiang (xinhaoji)
Yukun Jiang (yukunj)

Performance data and configurations

Database choosed (MySQL/HBase): MySQL
Number and types of instances: Number of instances: 1 master and 7 workers. All of them are m6g.large.
Cost per hour of the entire system: 0.077 * 8 (8 m6g.large instances)+ (32 G * 8 * 0.1 * G / month / 30 days / 24 hours (each instance has disk with 32 G)) + 0.0225 (network load balancer) = 0.616 + 0.0356 + 0.0225 = 0.6741
(Cost includes on-demand EC2, EBS and ELB costs. Ignore S3, EMR software addition, Network and Disk I/O costs)

Requests Per Second (RPS) of your <u>best</u> submission:

	M1	M2	M3
score	**10.00**	**50.00**	
submission id	**689094**	**696754**	
throughput	**86177.79**	**63374.51**	
latency	**5.55**	**6.67**	
correctness	**100.00**	**99.86**	
error	**0.00**	**0.00**	

Rubric

- Each unanswered bullet point = -4%
- Each unsatisfactory answer = -2%

- Use the report as a record of your progress, and then condense it before submitting it.
- If you write down an observation, we also expect you to try and explain it.
- Questions ending with "Why?" need evidence (not just logic).
- Always use your own words (paraphrase); we will check for plagiarism.

Task 1: Web-tier

Question 1: Comparing web frameworks

Microservice 1 and 2 does not involve database access and is a good opportunity for you to compare and choose a suitable web framework for future queries. Please try a couple of combinations of programming and web frameworks, and answer the questions below. To save cost during your comparison, we recommend that you deploy your testing code to a single EC2 virtual machine instead of a Kubernetes cluster.

- You are free to pick either M1 and M2 (or both) to compare web frameworks. Which Microservice did you choose and why?
  We choose M1 service to compare web frameworks, as by observation we find that M1 is less computationally intensive than M2, therefore more likely to reveal the inherent speed of receiving and responding back to the reference site of the web frameworks.
- What frameworks have you tried? How did each framework perform? Please include instance (or cluster) configuration, latency and RPS of each framework you tried.
  We used Spring Boot and Vert.X for web frameworks comparison. In a nutshell, Vertx is much more lightweight and faster than Spring Boot. We deployed 6 M6.large machines within a Kubernetes cluster. For the full 600 seconds M1 service test, we achieve the following performance:

	Effective Throughput	**Latency**	**Correctness**
Spring Boot	23789.54	80.76	100%
Vert.X	86177.79	9.65	100%

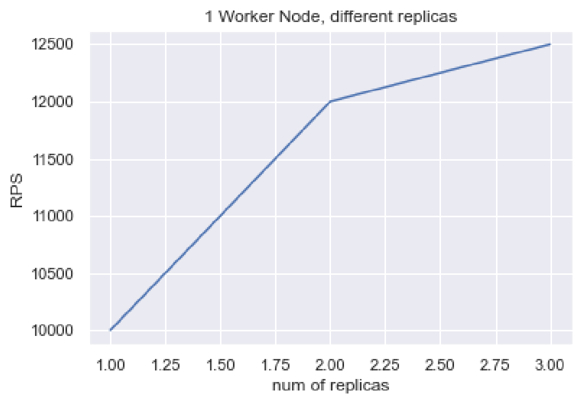
- For the two frameworks with the highest RPS, please list and compare their functionality, configuration, and deployment.
  The top two performing frameworks are Spring Boot and Vert.X. Spring Boot has the most wide-covered functionality in the industry and it's highly recognized. However, it's too weighted and therefore less performant in our service cases. On the other hand, Vert.X is said to be the top one of quick and lightweight web frameworks, and indeed it proves this in our testing performance. It's an asynchronous handler that after receiving a HTTP request, it will wake up and send it back after processing the request.
- Which one would you like to choose for other queries? What are the most important differentiating factors?
  We will go with the Vert.X since it meets the performance requirements and the services we are supporting are not that difficult to require the power of Spring Boot.

- Did you have to do any trade-offs while picking one of the multiple frameworks you selected?
  Yes. Spring Boot is more easy to use because the underlying code and structure takes care of a lot of things for us. We could achieve all functionalities within a few lines of code. But this is relatively slower than Vert.X. For Vert.X, we have to take care of parsing the HTTP request and writing the response by ourselves, but it has the best performance and that's what we are looking after.

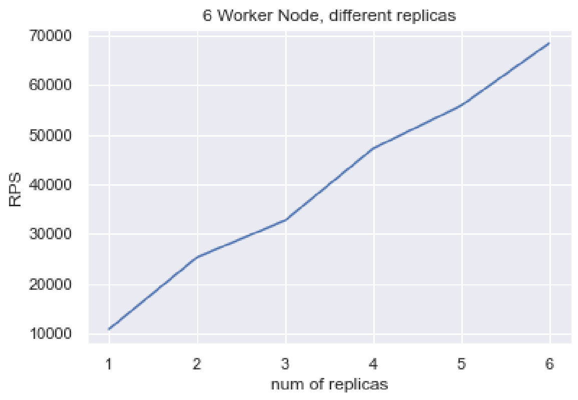
Question 2: Load balancing and scaling

Our target RPS for Microservice 1 and 2 are 65k and 25k respectively, which can be hard to reach with a single pod. We have learned about Kubernetes and ELBs in the individual projects, so let us explore the knowledge and skills you have developed here.

- Discuss load-balancing in general, why it matters in the cloud, and how it is applicable to your web-tier.
  Load balancing is the service/proxy that after receiving the HTTP requests from outside, it redirects to the specific responsible worker pods for that request. It's very important as a single web might be able to handle large numbers of service types. In our web-tier, we are supporting 3 kinds of services and we need to distribute different requests to where it belongs to.
- Provision a cluster with only one worker node and deploy your Microservice 1 using a Kubernetes deployment with one replica. Provision a Network Load Balancer by using Kubernetes service resource. Make a 120 second submission to the Sail() Platform and record the RPS. Add one replica to your deployment at a time, while keeping your worker node count to only 1. Wait until the target group shows the new pod as healthy, make another submission to the Sail() Platform and record the RPS. Repeat this process for 2 times, ending with 3 replicas running on 1 worker node.
  As the number of replicas increases, does the RPS grow accordingly? Please show the graph with the data points you just collected and write down a possible explanation.
  We observe that, using a Network Load Balancer, when deploying only one worker node, as the number of replicated pods increases, the RPS doesn't really scale up and it saturates. We guess it's because the worker machine is overwhelmed and it's CPU-intensive.



- Provision a cluster with as many worker nodes as possible while staying under the $0.70/h submission budget. Deploy your Microservice 1 using a Kubernetes deployment with only one replica and provision a Network Load Balancer by using Kubernetes service resource. Make a 120 second submission to the Sail() Platform and record the RPS. Add one replica to your deployment at a time, wait until the target group shows the new replica as healthy, make another submission to the Sail() Platform and record the RPS. Repeat this process until your number of replicas reaches your number of the worker nodes.
  As the number of replicas increases, does the RPS grow accordingly? Please show the graph with the data points you just collected and write down a possible explanation.
  Yes, this time indeed when we have 6 worker machines, the RPS grows almost linearly as the number of replicate pods increases.



- How do you check which pod is running on which worker node? Give a specific command. How would you ensure that the workload is distributed evenly across your cluster?
  The command is to : kubectl get pods --all-namespaces -o wide --field-selector spec.nodeName=<node>
  To ensure that the workload is distributed evenly across our cluster, we can set the desired CPU usage of a single pod on a core. By doing so, pods will be distributed evenly across the cluster and machines.
- Compare the difference between Application Load Balancer and Network Load Balancer. For each type, use the best cluster configuration you can have under the $0.70/h submission budget. Use the ingress resource to provision an ALB and service resource for NLB. Wait until the load balancers are in active state and the target groups show the pods as healthy. Submit the ALB/NLB endpoint to the Sail() Platform (M1, 600s) 6 times consecutively and record the RPS. Finish the following table.
  Note: Each type requires 6 submissions * 10 minutes consecutively, so it will take at least 1 hour, please plan your time accordingly.

The instance type of your master / worker nodes: M6.large
The number of instances in your cluster: 1 master node, 6 worker nodes

	Application Load Balancer	Network Load Balancer
Submission 1	13162.23	48358.98
Submission 2	14584.37	86177.79
Submission 3	15234.41	/
Submission 4	15368.44	/
Submission 5	16482.98	/
Submission 6	19560.17	/

We did not quite check ALB, and only tested classic LB and NLB. We did not know this question during phase 1, so we did not do the experiment. But luckily we get satisfying results from the Network Load Balancer already.

- We mentioned ALB warm-up in individual project P1. Explain the warm-up and why it is necessary based on the experiment above.
  By definition on the Wikipedia page, ALB warm-up is the process of initializing necessary services in full-scale to receive incoming requests. It's very important for our experiments here as we can see after each round of submission, the scores go up incrementally.
- We discussed scaling Kubernetes deployment in P2. Besides manually specifying the number of replicas in your deployment resource, describe one other technique that scales the web-tier of your microservice.
  Besides specifying the initial number of replicas in our deployment, we could also specify a max and min number of replicas and make it auto-scale-able, and the auto scaler will scale up or scale down the number of replicas in the cluster based on the recent workload it receives.

Question 3: Web-tier architecture

This question applies to all microservices. For each microservice we need a web-tier that interprets requests, fetches data, optionally retrieves data from the database and does processing to generate responses. How would you set up your web-tier for these purposes? You can use the following questions as a guide.

Keep in mind that, although you can test different microservices individually for now, in later phases your 3 microservices will need to be under the same public DNS and respond to requests at the same time. Make sure to consider this in your architecture and cluster design.

- Which web-tier architecture did you choose? Did you put pods of different microservices into the same worker node? If you choose MySQL as your storage-tier, did you put the pods of your web-tier of microservice 3 and MySQL pods into the same deployment?

We choose to deploy pods that can handle mixed requests on every pod, with one load balancer to handle all the requests. We choose Mysql as storage-tier, and we put the MySql pods and web-tier pods into the same deployment.

- What alternative architectures can you think of? What are their implications on performance and scalability and why?
  We could also split the pods into individually dealing with only 1 micro service. This would imply that we have a bad fault tolerance, i.e. if all pods corresponding to one service died, we lost that service. In contrast, in design 1 as illustrated above, each pod has the ability to respond to three services.
- Explain your choice of instance type and the number of worker nodes for your Kubernetes cluster.
  Since we need to fall down to the 0.7$/hour budget constraint, we need to calculate the budget carefully. We choose 6 m6.large as the worker nodes over 5 m5.large as the worker nodes, because they all have the same number of VCPUs and memory. M6.large is cheaper because it uses arm architecture over x86.

Question 4: Web-tier deployment orchestration

We know it takes dozens (or hundreds) of iterations to build a satisfactory system, and the deployment process takes a long time to complete. Setting servers/clusters up automatically saves developers from repetitive manual labor and reduces errors.

- We require you to use Terraform, Kubernetes and Helm to orchestrate the deployment of your web-tier. What steps were taken every time you deployed your system? Please include your Terraform script, Helm chart and Kubernetes manifest in your submission as per the requirements stated in the Phase 1 writeup. If you use kOps to provision your Kubernetes cluster, please also include the cluster definition YAML files in your submission. Starting from a clean local machine (with kops, kubectl and helm installed), we should end up with a Kubernetes cluster which has a functioning web-tier by applying your deployment steps.
  Yes indeed we used and included the automation scripts in our submission on github. Typically it takes 15 minutes to deploy the Kubernetes, and helm install all kinds of applications are really fast.
- If Terraform, Kubernetes and Helm are not required, what are some other ways to automate your web-tier deployment? Compare their advantages and disadvantages.
  By looking at the primers, we could also use Bash/Python/Ruby scripts, Ansible, Chef, Docker, Packer for web-tier deployment.
- How can you further automate the entire deployment process? If you implemented any other automation tools please describe them, and we would like to see your scripts in your code submission.
  No, we didn't use any other automation tools besides Kubernetes Kops and Terraform, helm. We believe these are all we need starting from a clean Kubernetes cluster.

Task 2: Extract Transform Load (ETL)

Question 5: Your ETL process

We have 1TB of raw data, which takes a non-trivial amount of time and money to process. It is strongly advised to test run and debug your E & T on a small portion of the data locally, and plan wisely before booting expensive clusters! Please briefly explain:

- The programming model/framework used for the ETL job and justification.
  We use PySpark on Azure HDInsight Spark Cluster for the ETL pipeline. As we believe PySpark would be very handy for iterative data processing steps, and we have quite a lot of budget left on every team member's account, we believe this would be a wise choice.
- The number and type of instances used during ETL and justification.
  Head Node: E8 V3 (8 Cores, 64 GB RAM) * 2
  Zoo Keeper: A2 V2 (2 Cores, 4 GB RAM) * 3
  Worker Node: E8 V3 (8 Cores, 64 GB RAM) * 10
  Our ETL design idea is quite meticulous and computationally expensive, therefore we need a rather large and powerful cluster to help do the computation within a reasonable amount of time.
- The execution time for the ETL process, excluding database load time.
  The overall ETL process takes around 1.5~2 hours, excluding database load time. This time includes data cleaning (filter out malformed tweets), transform the table into the dataframes that we want, outwrite to a .csv static file and collect and store it.
- The time spent loading your data into the database.
  We have three .csv distributed file folders to load into three MySQL database tables. The loading time for each table is around 7 minutes, therefore adding up to a total loading time of 20 minutes.
- The overall cost of the ETL process.
  Roughly estimating, we spent a total of around 150$ on the iterative ETL process due to many trials and errors, but this lies within our acceptable range.
- The number of incomplete ETL runs before your final run.
  We failed/unsatisfied with ETL runs for around 10 times before we made the final ETL run.
- The size of the resulting database and reasoning.
  The resulting database size is around 15~20GBs since the three static .csv file folders' size sum is around 20 GBs. This is reasonable because we did a lot of pre-computation during the ETL pipeline, a lot of scores are already ready for fetch in the database, hence we can dramatically reduce the amount of data storage, which would facilitate a faster data query time.
- The size and format of the backup.
  We use the cleaned data(filter out malformed tweets) and three final tables' .csv static file folders on AWS S3 bucket, so that every time we re-initialize a K8S cluster and database,

we could reload the data into that database. The total size of such backup is around 30GBs as appeared on the AWS S3 bucket info table.

- Discuss the difficulties encountered.
  At first, we didn't know how to load a file into a MySQL database. Furthermore, later we only know how to load a single file into a specific database table. Thanks to shell script, we find a way around to load every .csv file in a folder into the database in an automatic way.
- If you had multiple database schema iterations, did you have to rerun your ETL between schema iterations?
  It depends. Typically speaking, when you change the database schema, at least the last part of your ETL pipeline has to be rerun to get the ideal dataframe and store it as a static file for loading. However, we already plan the schema carefully and hopefully would be able to re-design the database schema and re-run the ETL process, which is time-consuming and constraint on budget.

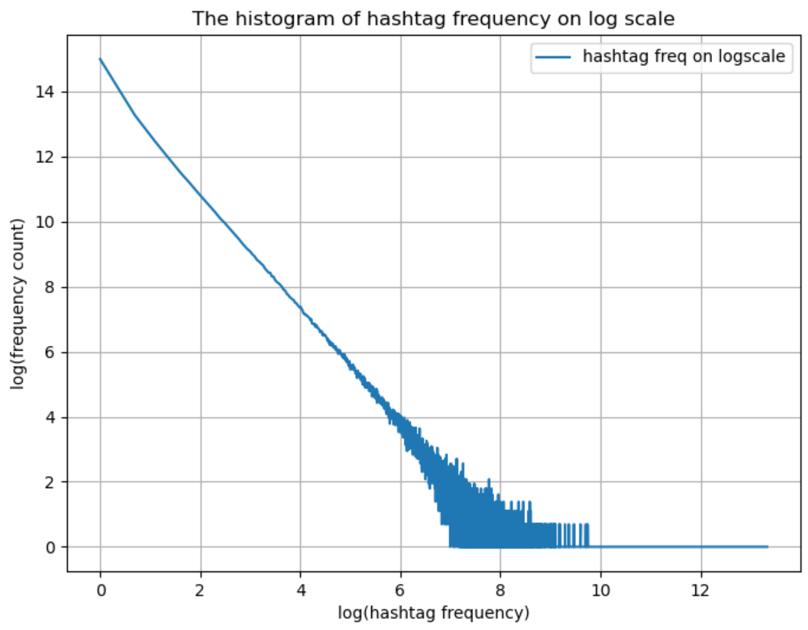
Question 6: ETL considerations

Historically, some teams ran out of the budget because they chose AWS for ETL and had multiple runs before they got their final dataset. Sometimes they simply used unnecessarily fancy (expensive) hardware. We believe you have compared and contrasted all the options and made smart decisions. Here are a few more questions about your understanding of the different choices:

- Which cloud service provider (AWS, GCP, Azure) and which service (EMR, Dataproc, HDInsight, Databricks, etc.) did you use and what was the difference?
  We mainly used Azure HDInsight Spark to perform our team's ETL pipeline. As is indicated in the writeup, it's dangerous to use AWS for the ETL as we have limited budget on it and we don't have too much credit in GCP, therefore Azure is a good choice.
- What are the most effective ways to speed up ETL?
  The most effective way to speed up ETL is to think clearly about any expensive join/groupby operations. Is it really necessary or is there a way around it? If you really have to do so, pre-partition the two dataframes into the same schema could greatly reduce the network shuffle write when joining. This is crucial from our observations.
- Besides Spark, list two other programming models you can use for ETL. Which approach would be more efficient and why? The term efficient can refer to many different aspects such as development time and actual run time, but as a result high efficiency should reduce the final ETL cost.
  Besides Spark, to do ETL we could also use the traditional MapReduce programming model or just do ETL in our favorite programming language. The drawbacks of Mapreduce is that everytime it has to store the intermediate result to HDFS on disk, which is expensive in an iterative process as we did in ETL. The drawbacks of using a programming language to do ETL is clear: we are mostly limited to a single machine's processing ability, which is inefficient given the 1TB dataset size.
- Did you encounter any error when running the ETL job in the provisioned cluster? If so, what are those errors? What did you do to investigate and recover from the error?

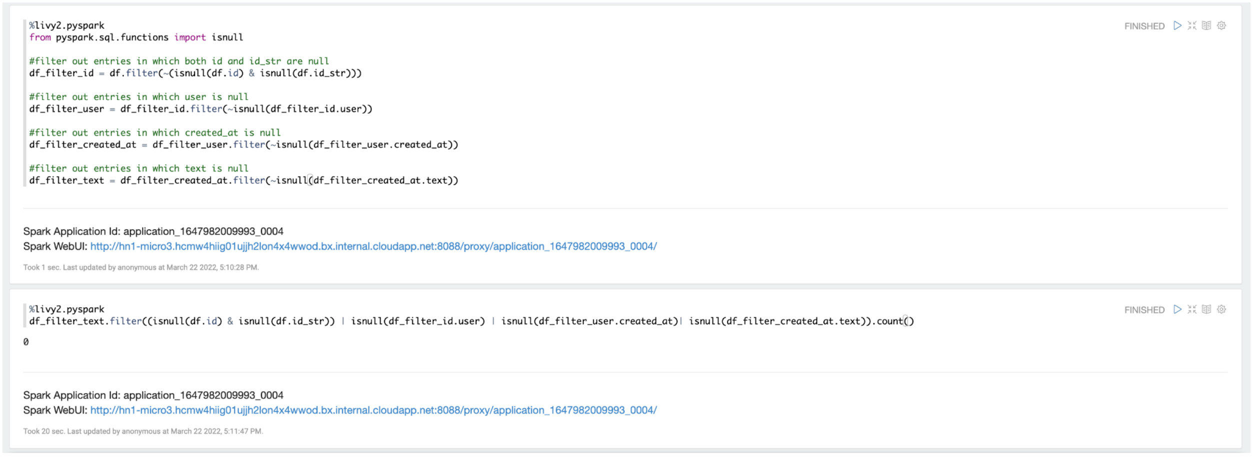
Yes, we did encounter quite a few errors. Firstly, we ran into a session timeout problem since the default Zeppelin will timeout after an hour, which is not enough for our ETL pipeline process, so we increased the time limit. Secondly, the memory setting for each executor and driver is very subtle. We ran into memory overdraft problems several times, and solved it after extensively reading related concepts and articles.

- How did you load data into your database? Are there other ways to do this, and what are the advantages of each method? Did you try some other ways to facilitate loading?
  The dataset after cleaning is in a distributed mode, i.e. it's scattered in a folder with up to 1000 partitions. We wrote a shell script to load each partition into its belonging MySQL database table. We store the file as a .csv file and load it. As far as we know, we could also store the file as .tsf or .json file and load it into the database. Also, we could use Java/Python programming language to connect to the MySQL database and insert it into the database online dynamically. We choose the static .csv file storage because we want to persist the file and avoid future ETL re-engineering in Phase2 and Phase3. The cost for storing such a static file on AWS S3 bucket is a relatively inexpensive operation.
- Did you store any intermediate ETL results and why? If so, what data did you store and where did you store the data?
  We envision that the ETL process might take several rounds and want to avoid repeatedly doing the first step of filtering out malformed tweets. Therefore, we store a copy of cleaned-malformed-tweets on the AWS S3 bucket.
- We would like you to produce a histogram of hashtag frequency on **log** scale among all the valid tweets **without hashtag filtering**. Given this histogram, describe why we asked you to filter out the top hashtags when calculating the hashtag score. [Clarification: We simply need you to plot the hashtag frequencies, sorted on the x-axis by the frequency value on the y-axis]



The log-scale hashtag frequency plot is illustrated as above. We could observe that for those popular tags (whose appearing frequency is greater than *e^10*), their frequency value is very tiny, which means they take a minor portion of the whole hashtags set. Therefore, we can safely discard them at low risk. This will help to prevent data imbalance problems introduced by hot-key and improve the overall ETL pipeline efficiency.

- Before you run your Spark tasks in a cluster, it would be a good idea to apply some tests. You may choose to test the filter rules, or even test the entire process against a small dataset. You may choose to test against the mini dataset, or design your own dataset that contains interesting edge cases. **Please include a screenshot of the ETL test coverage report here.** (We don't have a strict coverage percentage you need to reach, but we do want to see your efforts in ETL testing.)
  Note: Don't forget to put your code for ETL testing under the **etl** folder in your team's github repo **master** branch.
  We first filter out tweets with null id/id_str, user, created_at, and text. After doing so, we counted the number of tweets violating the above rules, and we got zero such tweets.



```
%livy2.pyspark
from pyspark.sql.functions import isnull

#filter out entries in which both id and id_str are null
df_filter_id = df.filter(~(isnull(df.id) & isnull(df.id_str)))

#filter out entries in which user is null
df_filter_user = df_filter_id.filter(~isnull(df_filter_id.user))

#filter out entries in which created_at is null
df_filter_created_at = df_filter_user.filter(~isnull(df_filter_user.created_at))

#filter out entries in which text is null
df_filter_text = df_filter_created_at.filter(~isnull(df_filter_created_at.text))
```

Spark Application Id: application_1647982009993_0004
Spark WebUI: http://hn1-micro3.hcmw4hiig01ujjh2lon4x4wwod.bx.internal.cloudapp.net:8088/proxy/application_1647982009993_0004/

Took 1 sec. Last updated by anonymous at March 22 2022, 5:10:28 PM.

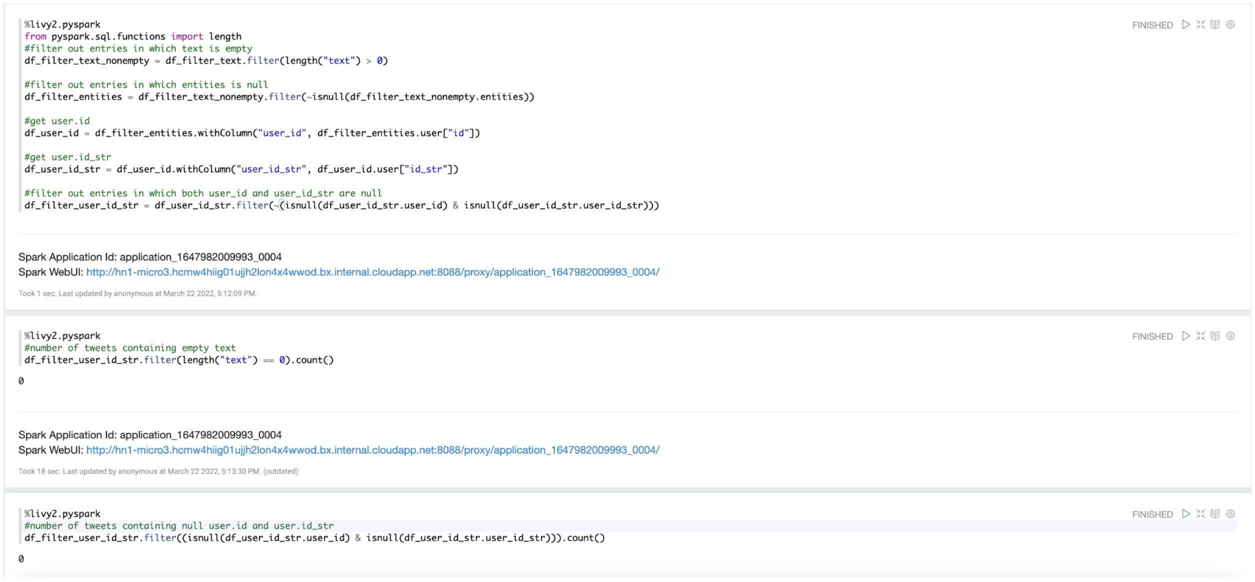
```
%livy2.pyspark
df_filter_text.filter((isnull(df.id) & isnull(df.id_str)) | isnull(df_filter_id.user) | isnull(df_filter_user.created_at)| isnull(df_filter_created_at.text)).count()

0
```

Spark Application Id: application_1647982009993_0004
Spark WebUI: http://hn1-micro3.hcmw4hiig01ujjh2lon4x4wwod.bx.internal.cloudapp.net:8088/proxy/application_1647982009993_0004/

Took 20 sec. Last updated by anonymous at March 22 2022, 5:11:47 PM.

Then, we filter out tweets with text with zero length, and tweets without user.id/user.id_str. After this, we counted the number of tweets violating the above rules, and got zero such tweets.



```
%livy2.pyspark
from pyspark.sql.functions import length
#filter out entries in which text is empty
df_filter_text_nonempty = df_filter_text.filter(length("text") > 0)

#filter out entries in which entities is null
df_filter_entities = df_filter_text_nonempty.filter(~isnull(df_filter_text_nonempty.entities))

#get user.id
df_user_id = df_filter_entities.withColumn("user_id", df_filter_entities.user["id"])

#get user.id_str
df_user_id_str = df_user_id.withColumn("user_id_str", df_user_id.user["id_str"])

#filter out entries in which both user_id and user_id_str are null
df_filter_user_id_str = df_user_id_str.filter(~(isnull(df_user_id_str.user_id) & isnull(df_user_id_str.user_id_str)))
```

Spark Application Id: application_1647982009993_0004
Spark WebUI: http://hn1-micro3.hcmw4hiig01ujjh2lon4x4wwod.bx.internal.cloudapp.net:8088/proxy/application_1647982009993_0004/

Took 1 sec. Last updated by anonymous at March 22 2022, 5:12:09 PM.

```
%livy2.pyspark
#number of tweets containing empty text
df_filter_user_id_str.filter(length("text") == 0).count()
```

```
0
```

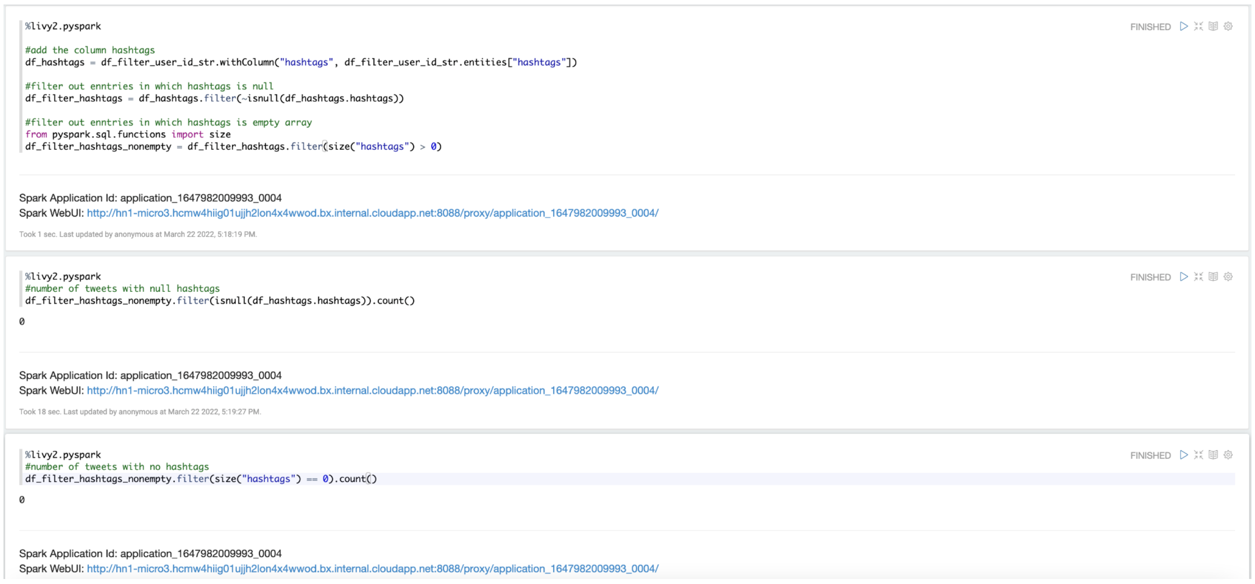
Spark Application Id: application_1647982009993_0004
Spark WebUI: http://hn1-micro3.hcmw4hiig01ujjh2lon4x4wwod.bx.internal.cloudapp.net:8088/proxy/application_1647982009993_0004/

Took 18 sec. Last updated by anonymous at March 22 2022, 5:13:30 PM. (outdated)

```
%livy2.pyspark
#number of tweets containing null user.id and user.id_str
df_filter_user_id_str.filter((isnull(df_user_id_str.user_id) & isnull(df_user_id_str.user_id_str))).count()
```

```
0
```

After this step, we filter out tweets with null hashtags or with zero hashtags. After this, we counted the number of tweets violating the above rules, and got zero such tweets.



```
%livy2.pyspark

#add the column hashtags
df_hashtags = df_filter_user_id_str.withColumn("hashtags", df_filter_user_id_str.entities["hashtags"])

#filter out enntries in which hashtags is null
df_filter_hashtags = df_hashtags.filter(~isnull(df_hashtags.hashtags))

#filter out enntries in which hashtags is empty array
from pyspark.sql.functions import size
df_filter_hashtags_nonempty = df_filter_hashtags.filter(size("hashtags") > 0)
```

Spark Application Id: application_1647982009993_0004
Spark WebUI: http://hn1-micro3.hcmw4hiig01ujjh2lon4x4wwod.bx.internal.cloudapp.net:8088/proxy/application_1647982009993_0004/

Took 1 sec. Last updated by anonymous at March 22 2022, 5:18:19 PM.

```
%livy2.pyspark
#number of tweets with null hashtags
df_filter_hashtags_nonempty.filter(isnull(df_hashtags.hashtags)).count()
```

```
0
```

Spark Application Id: application_1647982009993_0004
Spark WebUI: http://hn1-micro3.hcmw4hiig01ujjh2lon4x4wwod.bx.internal.cloudapp.net:8088/proxy/application_1647982009993_0004/

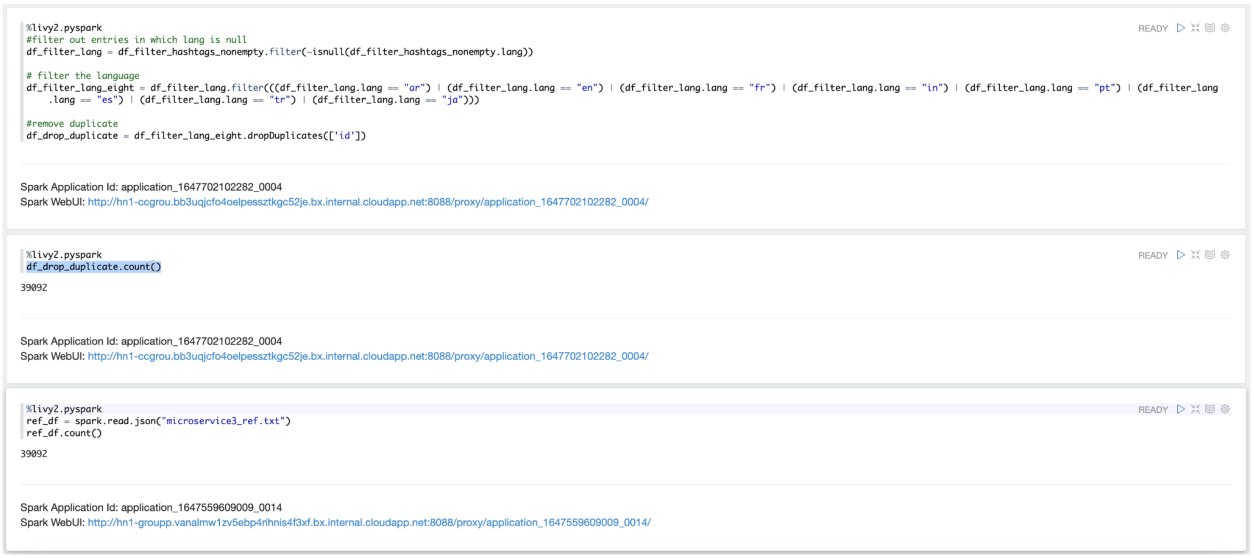
Took 18 sec. Last updated by anonymous at March 22 2022, 5:19:27 PM.

```
%livy2.pyspark
#number of tweets with no hashtags
df_filter_hashtags_nonempty.filter(size("hashtags") == 0).count()
```

```
0
```

Spark Application Id: application_1647982009993_0004
Spark WebUI: http://hn1-micro3.hcmw4hiig01ujjh2lon4x4wwod.bx.internal.cloudapp.net:8088/proxy/application_1647982009993_0004/

Finally, we keep tweets with correct language, and remove duplicates. The final dataset contains 39092 tweets, which matches the reference dataset.


```
%livy2.pyspark
#filter out entries in which lang is null
df_filter_lang = df_filter_hashtags_nonempty.filter(~isnull(df_filter_hashtags_nonempty.lang))

# filter the language
df_filter_lang_eight = df_filter_lang.filter(((df_filter_lang.lang == "ar") | (df_filter_lang.lang == "en") | (df_filter_lang.lang == "fr") | (df_filter_lang.lang == "in") | (df_filter_lang.lang == "pt") | (df_filter_lang
    .lang == "es") | (df_filter_lang.lang == "tr") | (df_filter_lang.lang == "ja")))

#remove duplicate
df_drop_duplicate = df_filter_lang_eight.dropDuplicates(['id'])
```

Spark Application Id: application_1647702102282_0004
Spark WebUI: http://hn1-ccgrou.bb3uqjcfo4oelpessztkgc52je.bx.internal.cloudapp.net:8088/proxy/application_1647702102282_0004/

```
%livy2.pyspark
df_drop_duplicate.count()
```

```
39092
```

Spark Application Id: application_1647702102282_0004
Spark WebUI: http://hn1-ccgrou.bb3uqjcfo4oelpessztkgc52je.bx.internal.cloudapp.net:8088/proxy/application_1647702102282_0004/

```
%livy2.pyspark
ref_df = spark.read.json("microservice3_ref.txt")
ref_df.count()
```

```
39092
```

Spark Application Id: application_1647559609009_0014
Spark WebUI: http://hn1-groupp.vanalmw1zv5ebp4rihnis4f3xf.bx.internal.cloudapp.net:8088/proxy/application_1647559609009_0014/

Task 3: Databases (MySQL or HBase)

Question 7: Database system

For Microservice 3, you are free to choose between MySQL or HBase. In the checkpoint report, we asked you to explore the differences between MySQL and HBase.

- Which database system did your team choose eventually, and why?
  We choose MySQL database over HBase, i.e. relational database over NoSQL. This is because by examining the data required for micro service 3, it naturally comes out of a "relational" schema, specified by user interaction and scores. Therefore, we believe if we deploy MySQL database system, it will be more performant compared to HBase.

Question 8: Schema

We first extract and transform the raw data and then store it in databases in a good format. Various schemas can result in a very large difference in performance! In Phase 1 we only ask you to reach 6% of the target RPS (600 out of 10000) of Microservice 3 for receiving checkpoint score. If you achieve 2000 rps, you will get the M3 Early Bird Bonus. These targets can be achieved with a basic but functional schema. In Phase 2 your Microservice 3 will need to reach 100% of the target (10,000) RPS which would require a more efficient schema.

When you are optimizing your schema in Phase 2, you will want to look at different metrics to understand the potential bottlenecks and come up with a better solution (before trying to tune parameters!). These iterations take a lot of time and a lot of experimentation so keep this in mind while designing your schema in Phase 1.

- What is your schema for Microservice 3 in Phase 1? Did you already have multiple iterations of schemas? If so please describe the previous schemas you have designed and explain why you decide to iterate to a new schema.

Since we found that that the interaction score and hashtag score could be precomputed, we design our schema into 3 tables:

- Table1: userA - userB - product of interaction & hashtag score - latest reply tweet - latest retweet tweet - latest both tweet
- Table2: user - screen name - description
- Table3: sender - receiver- tweet content - tweet hashtags - is_reply

We don't have iterations of the schema so far, as we thoughtfully designed it in the first place and believe it will surely excel.

- List 2 potential optimization that can be applied to your schemas. How might they improve your performance?
- Firstly, the match score for phase and hashtag is now dynamically computed, i.e. we load all related tweets from the MySQL database into Java and do matching score calculations in the Java Program. However, it's also possible to pre-classify the phase and hashtag in each tweet and have them ready at hand. This will improve the query and response performance.
- Secondly, currently when we rank and find the to-be-returned users, we have to do another round of query in the database to find their latest screen name and description. This might pose a performance bottleneck on the MySQL database. It's possible for us to attach the user information table (table2) to the last columns of table1 and save query speed at the expense of more space occupation.

Question 9: Optimizations and Tweaks

Frameworks and databases do not come in the best shape out-of-the-box. Each of them has tens of parameters to tune. Once you are satisfied with the web framework, you can start learning about the configuration parameters to see which ones might be the most relevant and gather data and evidence with each individual optimization. Although you might not have arrived at an optimized database schema yet, feel free to learn about tunable parameters in your database system. No need to feel obliged to test all of them now. In future phases, you will continue to try to improve the performance of your microservices.

Hint for Phase 2: A good schema will easily double or even triple the target RPS of your Microservice 3 without any parameter tuning. It is advised to first focus on improving your schema and get an okay performance with your best cluster configuration. If you are 10 times (an order of magnitude) away from the target, then it is very unlikely to make up the difference with parameter tunings.

- List as many possible items to tweak as you can find that might impact performance, in terms of: Kubernetes config, web framework, your code, DBs, DB connectors, OS, etc. Choose at least three optimizations, apply them one at a time and show a table with the microservice you are optimizing, the optimization you have applied and the RPS before and after applying it.

	Improvement Plan	RPS before	RPS after

Micro1	Early stopping once the validation failed	~35000	~63000
Micro1	Declare the QRCoder solver as an instance in the webframe instead of calling it as a static method every time	~63000	~68000
Micro2	Parallelize the validation and new hash computation	~73000	~86000

Beside, we also consider the followings:
Kubernetes cluster's size and machine types, how many Vcpus and RAM to give to each worker node.
The web application framework inherent performance.
Our code' s utilization of the underlying hardware and space usage efficiency.

- For every tweak you listed above, try to explain how it contributes to performance.
  For 1, once we find a small error in the QR code's matrix, we could directly return the invalid response instead of carrying out the computation to the end. There are quite a number of invalid cases in the test.
  For 2, the static method is not as efficient as directly initializing a new class instance in the web server Vertx.
  For 3, we don't have to wait all the way till the validation of block chains is finished. We parallelize the computation of hash code. This greatly increases the CPU utilization rates.
- Which optimizations only exist in one type of DB and why? How can you simulate that optimization in the other (or if you cannot, why not)?
  Creating Index only exists in relational databases like MySQL. To simulate a similar effect in NoSQL like HBase, we could declare smart rowkey and column families instead.

Question 10: Storage-tier deployment orchestration

In addition to your web-tier deployment orchestration, it is equally important to have automation and orchestration for your storage-tier deployment in order to save labor time and avoid potential human errors during your deployment.

- We require you to use Terraform, Kubernetes and Helm to orchestrate the deployment of your storage-tier. What steps were taken every time you deployed your system? If you use MySQL as your storage-tier, include your Helm chart and Kubernetes manifest in your code submission on GitHub. If you use HBase, include your Terraform script which provisions EMR in your submission. Starting with a Kubernetes cluster that has your

web-tier running, we should be able to have a functional storage-tier that is reachable from your web-tier after running your deployment scripts.
Mostly, the most time-consuming step is to deploy a Kubernetes cluster. For helm install applications, it's relatively fast. Also, to create MySQL db pods and mount snapshot volume is also a relatively expensive time to wait for. We include the related relevant scripts in our github submission.

- Making snapshots of your database can save time importing your ETL results into the database if you need to re-deploy your storage-tier. What are some methods to make snapshots of your database? Did you implement any of them? If so, how much time have you saved each time you restore the snapshot, compared to having to import ETL results again?
We first deploy temporary MySQL pods and declare a clean 50GB volume, attach to the MySQL pods and load data into it from the .csv file. After that, we take a snapshot of the state of this database volume. Therefore, later on when we re-create the MySQL database, we could mount the snapshot instead of reloading the ETL .csv file into the database. This saves the time from half an hour to just a few minutes every time re-creating the MySQL database.

Task 4: Site reliability

Question 11: Monitoring

Once the service is set up and running, it begins to generate a large amount of status data. They help us monitor if the system is operational, and also give us insights into its performance.

- What CloudWatch metrics are useful for monitoring the health of your system? What metrics help you understand performance?
Status check failure can help us monitor the health of our system.
CPU utilization and Network out helps us understand performance.
- What statistics can you collect if you have SSH access into the VM? What tools did you use? What did they tell you?
Top command can help us. We could check things like cpu utilization, such that we can know if we have fully utilized our machine.
- What statistics can you collect if you have kubectl access, but not SSH access into the nodes? What tools did you use? What did they tell you?
We can check the status of each pod by using things like kubectl get pods. In that way, we can know if our pods are healthy.
- How do you monitor the status of your database? What interesting facts did you observe?
We can use "mysqladmin -u root status " and "kubectl describe pods" to monitor MySQL status. One interesting fact we observed is that there are barely slow queries in MySql.

Question 12: Profiling

We expect you to know the end-to-end latency of your system. This helps us to focus on optimizing the parts of the system that contribute to most of the latency.

- Given a typical request-response for Microservice 3, for each latency below, think about if you can measure it, how you can measure it, and write down the tools/techniques that can be used to measure it:
    - a Load Generator to Load Balancer: We estimate this latency by responding to empty strings without cluster and computation.
    - b Load Balancer to Worker Node: We measure the latency without cluster, and with a cluster and load balancer. We approximate it by calculating the difference between those two.
    - c Worker Node to Web-tier Pod
    - d Parsing request: time the functions in the web server.
    - e Web Service to DB:measure query time on the web server.
    - f At DB (execution): measure DB execution time on mysql server.
    - g DB to Web Service:measure query time on the web server.
    - h Parsing DB response:time the functions in the web server.
    - i Web-tier Pod to Worker Node
    - j Worker Node to Load Balancer: same as part b. We can also estimate the bottleneck by monitoring the network latency and CPU utilization on AWS cloudWatch.
    - k Load Balancer to Load Generator, same as part a.
- For each part above, think about whether you can improve the latency or not, and list possible ways to reduce the latency.
    1. We can improve latency of DB query by pooling of DB connection, and employ async non blocking IO to improve performance.
    2. We can also improve DB queries by preparing query statements.
    3. We can improve the latency of load balancers by using faster load balancers.
    4. We can decrease web server latency by caching results.

Task 5: General questions

Question 13: Scalability

In this phase, we serve read-only requests from tens of GBs of processed data. Remember this is just an infinitesimal slice of all the user-generated content on Twitter. Can your servers provide the same quality of service when the amount of data grows? What if we realistically allow updates to tweets? These are good questions to discuss after successfully finishing this phase. It is okay if you answer no, but we do want you to think about them and share your understanding.

- Would your design work as well if the quantity of data was 10 times larger? What about 100x? Why or why not? (Assume you get a proportional amount of machines.)

Our design will work less fast but not linearly. Most of the latency comes from database query, and the query latency grows in Log(n) with size of data. Therefore, with 100x more data, our program will theoretically work 2x slower, which is a reasonable latency.

- Would your design work if your web service also implemented insert/update (PUT) requests? Why or why not?
  This will provoke more challenges to our database because wil indexing, the writing will be much slower. We tend to devise a new design if writing/updating is included.

Question 14: Postmortem

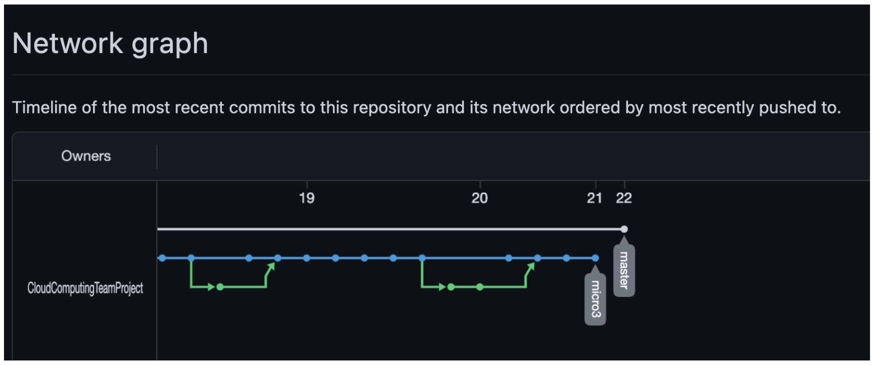
- How did you set up your local development environment? Did you install the databases locally on your machines to experiment? Did you test all programs before running them on your Kubernetes cluster on the cloud?
  We basically set up the same development environment locally as that on the cloud. We installed the mysql server locally for easier tests. We also have tested most of not all of the programs locally before deploying on Kubernetes.
- Did you attempt to generate a load to test your system on your own? If so, how? And why?
  No, we didn't because logically a small set of test cases can effectively test the correctness of our program, and we tested the performance of our arche texture using the reference load generator because it's more efficient.
- Describe an alternative design to your system that you wish you had time or budget to try.
  An alternative design is more worker nodes with better computation optimization and larger memories if we have more budget.
  With more time, many more designs can be employed on web servers. For example, we can implement a Cache class in the web server.
- Which were the toughest roadblocks that your team faced in Phase 1?
  Our toughest roadblock was ETL because it was too time-consuming and Spark doesn't give any specific error message.
- Did you do something unique (any cool optimization/trick/hack) that you would like to share?
  We precomputed everything static in ETL and stored it in the database to minimize computation on the server.

Question 15: Non-technical

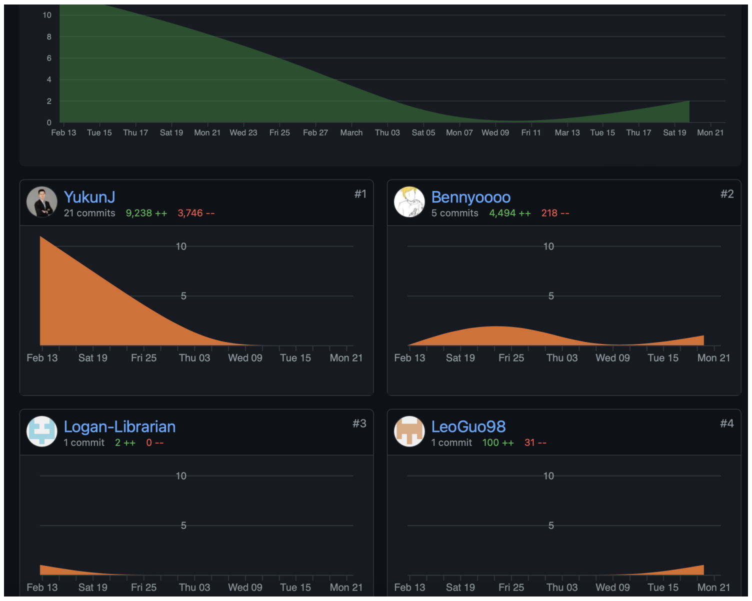
- How did you divide components in the system? How do the components interact? How did you partition the work and what was each team member's contribution?
  We divided the work into different parts of ETL, database deployment, and web server. The schema of the database will decide how we design ETL implementation and how web servers query the data. Therefore, we discuss and decide the data schema together first, and then work on each component parallely. Benny implemented a web server, Yukun deployed the database, and Leo was in charge of initial steps of ETL. Since ETL work is

heavier in micro3, we divided ETL work into different tasks, and we all contributed to this job.

- How did you test each component? Did you implement Unit Testing or Integration Testing?
  We tested ELT by comparing the results with smaller reference datasets at each critical step(cleaning, transformation, loading).
  We tested web servers by writing unit test cases for each sub-functionalities, and 10-20 end-to-end tests by comparing with reference servers.
- Did you like the Git workflow, code review, and internal testing requirements? Were they helpful to your development?
  We benefited a lot from Git workflow, code review, and internal testing process. These practices help us make more robust progress for each implementation.
- Please show Github stats to reflect the contribution of each team member. Specifically, include screenshots for each of the links below.
  1. https://github.com/CloudComputingTeamProject/<repo>/graphs/contributors
  2. https://github.com/CloudComputingTeamProject/<repo>/network


Network graph

Timeline of the most recent commits to this repository and its network ordered by most recently pushed to.


YukunJ #1
21 commits 9,238 ++ 3,746 --

Bennyoooo #2
5 commits 4,494 ++ 218 --

Logan-Librarian #3
1 commit 2 ++ 0 --

LeoGuo98 #4
1 commit 100 ++ 31 --

Yukun is the main “manager” of our github folder, since we are always working together, the workload is evenly distributed between team members and the git graph may be a bit inaccurate.

Task 6: Bonus

5% Microservice 2 Early Bird Bonus

Fill in the form if your team was able to achieve the Microservice 2 target by 03/06 and hence eligible for the 5% early bird bonus. Please push all the files you wrote to reach the target RPS and tag your git commit as "earlyBird-m2" so that we know you completed this bonus at that commit.

	M2
score	50
submission id	696754
throughput	63374.51
latency	6.67
correctness	99.86
error	0
date/time	2022-03-05 01:34:27

Penalty Waiver for Microservice 2 Correctness

Fill in the form if your team was able to make a 10-minute Microservice 2 submission with above 95% correctness by 03/06 and hence eligible for waiving one most significant penalty for each team member.

	M2
score	50
submission id	696754
throughput	63374.51
latency	6.67
correctness	99.86
error	0
date/time	2022-03-05 01:34:27

5% Microservice 3 Early Bird Bonus

Fill in the form if your team was able to achieve the Microservice 3 phase 1 target (2000 RPS) by 03/20 and hence eligible for the 5% early bird bonus. Please push all the files you wrote to reach the target RPS and tag your git commit as “earlyBird-m3” so that we know you completed this bonus at that commit.

	M3
score	
submission id	
throughput	
latency	
correctness	
error	
date/time	

Penalty Waiver for Microservice 3 Correctness

Fill in the form if your team was able to make a 10-minute Microservice 3 submission with above 95% correctness by 03/20 and hence eligible for waiving one most significant penalty for each team member.

	M3
score	
submission id	
throughput	
latency	
correctness	
error	
date/time	