Team Project Phase 2 Report

Summary

Team name:
Members:

Name	Andrew ID
Kewen Zhao	Kewenz
Yu Wei	yuwei2
Zhitong Guo	zhitongg

Please color your responses as red and upload the report in PDF format.

Live Test Performance & Configurations

Number and types of instances:
Cost per hour of the entire system:
(Cost includes on-demand EC2, EBS, and ELB costs. Ignore S3, Network, and Disk I/O costs)

Your team's overall rank on the live test scoreboard:
Live test submission details:

	Blockchain	QR Code	Twitter	Mixed Test (Blockchain)	Mixed Test (QR Code)	Mixed Test (Twitter)
score	20	12	15.56	10	8	10
submission id	940915	940916	940917	940918	940918	940918
throughput	82475.67	85265.13	7994.56	14417.04	29152.28	5847.67
latency	6.17	6.22	83.27	8.82	8.64	9.20
correctness	92.48	100.00	97.29	92.47	100.00	97.29
error	0.00	0.00	0.00	0.00	0.00	0.00

Rubric

- Each unanswered bullet point = -4%

- Each unsatisfactory answer = -2%
- Optional Questions = +4%

Guidelines

- Use the report as a record of your progress, and then condense it before submitting it.
- When documenting an observation, we expect you to also provide an explanation for it.
- Questions ending with "Why?" require evidence, not just reasoning.
- It is essential to express ideas in your own words, through paraphrasing. Please note that we will be checking for instances of plagiarism.

Task 1: Improvements of Microservices

Question 1: Cluster architecture

You are allowed to use several types of EC2 instances, and you are free to distribute the workload across your cluster in any way you'd like (for example, hosting MySQL only on some worker nodes, or having worker nodes of different sizes). When you try to improve the performance of your microservices, it might be helpful to think of how you make use of the available resources within the provided budget.

- List at least two reasonable varieties of cluster architectures. (e.g., draw schematics to show which pieces are involved in serving a request).
- In twitter and qrcode tasks, we need to deploy two services(mysql and auth). There are two possible ways
- 1. Decoupled architecture: Separate two services into different pods: This approach decouple individual services, allowing for better scalability and independence.
- 2. Monolithic architecture: Putting both services into one pods: This approach allows communication between services faster since they lies in the same pod
- Discuss how your design choices mentioned above affect performance, and why.
- Monolithic architecture has less communication overhead because the request between services remains local, no DNS routing is needed.
- Decoupled architecture: Separate two services into different pods: This approach decouple individual services, allowing for better scalability and independence.
- Describe your final choice of instances and architecture for your microservices. If it is different from what you had in Phase 1, describe whether the performance improved and why.
- We adopted Monolithic architecture because both services' workloads are balanced: each request to the web-tier triggers exactly one request to the supporting service, therefore we don't need to worry about scalability due to imbalance in workload. Also Monolithic architecture is faster.

Question 2: Database and schema

If you want to make Twitter Service faster, the first thing to look at is the database because the amount of computation at the web tier is not as much as in Blockchain Service and QR Code Service. As you optimize your Twitter Service, you will find out that various schemas can result in a very large difference in performance! Also, you might want to look at different metrics to understand the bottleneck and think of what to optimize.

- What schema did you use for Twitter in Phase 1? Did you change it in Phase 2? If so, please discuss how and why you changed it, and how did the new schema affect performance?
- We did not make huge changes to the schema we are using as we have carefully designed our database and query processes during the ETL process. In Phase 1 and Phase 2, we are using basically the same schema listed below.
- We have two tables “combinedScores” and “screen_name_and_description_latest”
- For “combinedScore” , we calculated the hashtag scores and interaction scores between any two users beforehand and stored these scores in the table we have the following columns:
- from_user_id, to_user_id: the user pair
- hashtag_score, interaction_score: the scores calculated during ETL process
- reply_aggregated_texts, reply_aggregated_hashtags, retweet_aggregated_texts, retweet_aggregated hashtags: resources needed to calculate the keyword score
- latest_tweet_text: latest contact tweet of the type specified in the query and with the user in the query.
- For “screen_name_and_descrpition_latest”, we just save all the personal information needed in the web response: latest_screen_name, description and their corresponding user_id
- Compare any two schemas for Twitter. Try to explain why one schema is more performant than another.
- Since we did not make significant changes to the schema. We just describe how we design the schema to make this web service performant.
- Firstly, we calculated all the user pairs’ hashtag scores and interaction scores ahead of time so that we do not need time for recalculation simply by adding two columns to the table.
- Secondly, we split the user information and interactions between each other to two independent tables so as to reduce the duplication of information and avoid the use of join operations as less frequent as possible.
- Explain briefly the theory behind at least 3 performance optimization techniques for databases in general. How are each of these optimizations implemented in your storage tier?
- Firstly, datatype choices:
- With respect to the ids of users which we are using for filtering queries, we choose BIGINT as our choice because we need to speed up the query and an integer primary key will always be faster for sorting and SELECTs than an alphanumeric primary key when the data is more than 4 characters long.

- Secondly, indexing:
- As we only consider query as our only operation in our web server, so we do not need to worry that indexing will slow down the insert and delete operations. And we only used user_id as our query filter, so we only created 1 index on the user_id column of each table.
- Thirdly, denormalization:
- As we only considered the time efficiency for our project. Denormalization is another trick we used. We calculated each user pair's interaction and put the scores of these pairs together in one table to avoid join operations and speed up queries.

Question 3: Your ETL process

It's highly likely that you need to re-run your ETL process in Phase 2 each time you implement a new database schema. You might even find it necessary to re-design your ETL pipeline if your previous one is insufficient for your needs. For the following bullet points, please briefly explain:

- The programming model/framework used for the ETL job and justification

Apache Databricks, which has good UI, and is able to get jobs finished very quickly

- The number and type of instances used during ETL and justification

5 worker nodes and 1 master node; we have tested on 2 worker nodes and 1 master node before, and run into many incomplete executions due to limitations of memory.

- The execution time for the ETL process, excluding database load time

3 hours

- The time spent loading your data into the database

~20 minutes (loading dump)

- The overall cost of the ETL process

~200$

- The number of incomplete ETL runs before your final run

~10 runs

- The size of the resulting database and reasoning

19 GB, which is huge table containing the latest user info and latest interaction tweets of a user pair

- The size and format of the backup

19 GB mysql dump in .sql extension

- Discuss the difficulties encountered

We tried using Zeppelin notebook, which is very slow and always run out of memory, and we also had confusions about the definitions of interaction tweets, causing us to ignore the user status in the retweet_status field.

- If you had multiple database schema iterations, did you have to rerun your whole ETL pipeline between schema iterations?

We did rerun once or twice, but most of the iterations were done on smaller test databases.

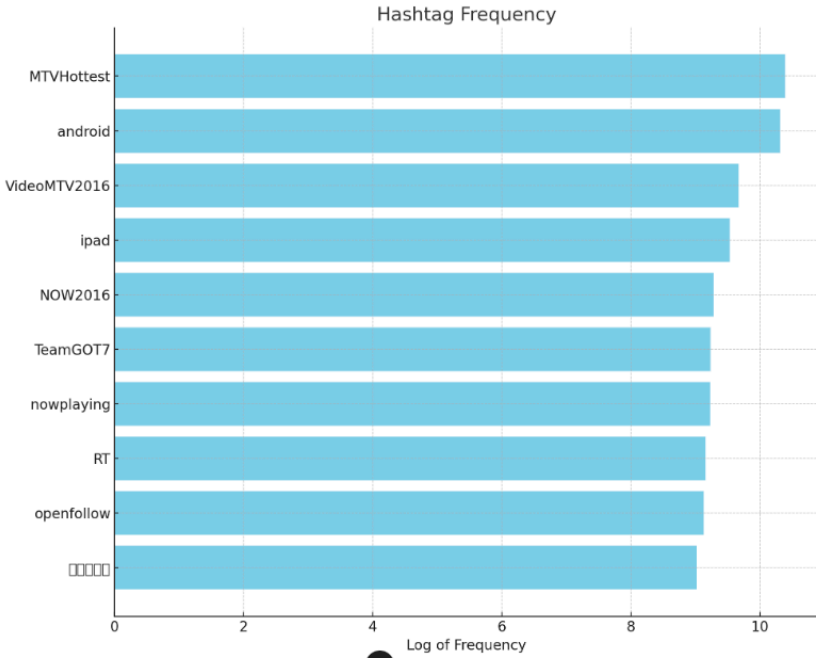
- How did you load data into your database? Are there other ways to do this, and what are the advantages of each method? Did you try any of these alternate methods of loading?

We loaded the data into the database by forwarding the port to localhost, and then mysql load the dumpfile. There are other ways, such as scp the dumpfile to the database then load, but it is very slow and requires probably twice the size of dump file for the disk space.

- Did you store any intermediate ETL results and why? If so, what data did you store and where did you store the data?

No, we did not.

- We would like you to produce a histogram of hashtag frequency on a **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]



- 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 compared the counts of the etl result of the first partition of the dataset, with the counts of the example result provided in the writeup, and discovered that they are the same, meaning we have the correct filters.

```
sampleDf_2mb.count()
```

```
res21: Long = 6500
```

Took 1 hrs 6 min 56 sec. Last updated by anonymous at March 14 2024, 6:24:57 PM.

```
import org.apache.spark.sql.functions._
import org.apache.spark.sql.SparkSession

val spark = SparkSession.builder.appName("Filter Tweets").getOrCreate()
import spark.implicits._

val tweets = sampleDf_2mb.cache

val validTweets = tweets.filter(
    ($"id".isNotNull || $"id_str".isNotNull) &&
    ($"user.id".isNotNull || $"user.id_str".isNotNull) &&
    $"created_at".isNotNull &&
    $"text".isNotNull && $"text" =!= "" &&
    $"entities.hashtags".isNotNull && size($"entities.hashtags") =!= 0 &&
    array_contains(array(lit("ar"), lit("en"), lit("fr"), lit("in"), lit("pt"), lit("es"), lit("tr"), lit("ja")), $"lang")
).cache

val uniqueTweets = validTweets.dropDuplicates("id_str").cache
```

```
import org.apache.spark.sql.functions._
import org.apache.spark.sql.SparkSession
spark: org.apache.spark.sql.SparkSession = org.apache.spark.sql.SparkSession@7ea17d2e
import spark.implicits._
tweets: sampleDf_2mb.type = [contributors: string, coordinates: struct<coordinates: array<double>, type: string> ... 36 more fields]
validTweets: org.apache.spark.sql.Dataset[org.apache.spark.sql.Row] = [contributors: string, coordinates: struct<coordinates: array<double>, type: string> ... 36 more fields]
uniqueTweets: org.apache.spark.sql.Dataset[org.apache.spark.sql.Row] = [contributors: string, coordinates: struct<coordinates: array<double>, type: string> ... 36 more fields]
```

Took 1 hrs 6 min 34 sec. Last updated by anonymous at March 14 2024, 6:24:58 PM.

```
validTweets.count()
```

```
res22: Long = 773
```

Took 9 sec. Last updated by anonymous at March 14 2024, 6:25:06 PM.

Question 4: 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 schema, you can start learning about the configuration parameters to see which ones might be most relevant and gather data and evidence with each individual optimization. You have probably tried a few in Phase 1. To perform even better, you probably want to continue with your experimentation.

Hint: A good schema will easily double or even triple the target RPS of your Twitter Service 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 configure or tweak that might impact performance, in terms of web framework, your program, DBs, DB connectors, OS, etc.

- 1. One Table: We only stored the latest contact tweets for each user pair, and we calculated the hashtag score and interaction score before hand, to make the database compact and the web tier straight forward.
- 2. Two Table: the first table stores pairwise information between users, such as interaction scores, tweet text, etc. The second table stores single user information, namely user description and screen name. The second table is in Conjunctive normal form(CNF) while the first table isn't

- 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.
- One Table: RPS 7800.32
- Two Table: RPS 672.47
- For every configuration parameter or tweak, try to explain how it contributes to performance.

- One Table approach is significantly faster because we only need to make one SQL query for each request. Two Tables approach need to make a request for each contact user to get their screen name and description, which is way slower

Task 2: Development and Operations

Question 5: Web-tier orchestration development

We know it takes dozens or hundreds of iterations to build a satisfactory system, and the deployment process takes a long time to complete. We asked about the automated deployment process in the Phase 1 Final Report. Answer the following questions to let us know how your automation has changed in Phase 2.

- Did you improve your web-tier deployment process? What were the improvements or changes you made? Include your improved/changed Terraform scripts, Kubernetes manifest or Helm charts under the folders referring to Phase 1 folder structures.
- In the ig-nodes.yaml and deployment.yaml we increased the max/minsize of machine and replicas and changed the machineType from m7g.xlarge to m7.large to boost efficiency with little cost increase. Furthermore, we change the service type to NodePort so as to balance and route the load to different web services with better performance.

Question 6: 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. Answer the following questions to let us know how your storage-tier deployment orchestration has changed in Phase 2 as compared to Phase 1.

- Did you improve your storage-tier orchestration? What were the improvements or changes you made? Include your improved/changed Terraform scripts, Kubernetes manifest or Helm charts under the folders referring to Phase 1 folder structures.
- We created a mysql.yaml that automatically provision our mysql service. It utilize presistentVolumeClaim and volumeSnapshot to automatically mount saved data to mysql. They are under helm/twitter/templates

Question 7: Live test and site reliability

Phase 2 is evaluated differently than how we evaluated Phase 1. In Phase 2, you can make as many attempts as you want before the deadline. However, all these attempts do not contribute to your score. Your team's Phase 2 score is determined by the live test. It is a one-off test of your web service during a specified period of time, and so you will not want anything to suddenly fail; if you encounter failure, you (or some program/script) probably want to notice it immediately and respond!

- What CloudWatch metrics are useful for monitoring the health of your system? What metrics help you understand performance?

CPU utilization is what we mainly monitored, and it remained balanced but high activity. Also Request per second helps understanding the load pattern

- Which statistics did you collect when logged into the EC2 VM via SSH? Which tools did you use? What do the statistics tell you?
- Disk read/write speeds and queue lengths, to monitor if the disk I/O is becoming a bottleneck.
- Which statistics did you collect using kubectl? What do the statistics tell you?
- Pod CPU and memory usage via kubectl top pod, to identify resource-intensive pods.
- Certain pods were consistently using close to their CPU and memory limits, indicating they were sized too small for the workload.
- How did you monitor the status of your storage tier? What interesting facts did you observe?
- Monitored Persistent Volume (PV) and Persistent Volume Claim (PVC) statuses using kubectl get pv and kubectl get pvc.
- Some persistent volume claims were consistently at or near their capacity limits, suggesting a need for resizing or cleanup.
- How would your microservices continue to serve requests (possibly with slower response times) in the event of a system failure? Consider various scenarios, including program crashes, network outages, and even the termination of your virtual machines.
- Utilized Kubernetes liveness and readiness probes to detect and automatically restart failed containers.
- During the live test, did anything unexpected happen? If so, how big was the impact on your performance, and what was the reason? What did you do to resolve these issues? Did they affect your overall performance?
  Nothing unexpected, the performance was good

Task 3: General questions

Question 8: 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? Here are a few good questions to think about after successfully finishing this phase.

- 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.)
- No. Because the size of database instances needs to be increased through vertical scaling to handle higher loads or using horizontal scaling/ sharding to spread the load across multiple instances. Also, caching layers like Redis may be necessary to cache frequent queries.
- Would your design work if your web service also implemented insert/update (PUT) requests? Why or why not?
- No. As long as we take insert/update into consideration. Synchronization is a must for us to consider so as to keep the ACID quality of databases when it comes to multithreading. However, we did not implement synchronization, which makes our service not able to work when faced with PUT requests.

- Furthermore, as we are using denormalization and indexing, which will make insert/update(PUT) requests very inefficient. Our design for now will not get satisfactory performance.
- What kind of changes in your schema design or system architecture would help you serve insert/update requests? Are there any new optimizations that you can think of that you can leverage in this case?
- Firstly we have to use normalization to reduce redundant data by splitting the scores and resources to calculate them to different tables.
- Secondly, we cannot do the calculation of hashtag scores and interaction scores in the ETL process anymore as they are changing right now and we need to make two new tables for them to calculate and update the scores.

Question 9: 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 your programs before running them on your Kubernetes cluster on the cloud?
- For web tier and storage tier, we indeed set our local development environment. As ntex and axum are relatively lightweight web frameworks, we set up http server locally on our PC for correctness testing. We also have a small subset of data stored in the local mysql database to do simple tests. Thus we did test our programs for correctness before running them on Kubernetes clusters on the cloud.
- Did you attempt to generate a load to test your system on your own? If so, how? And why? (Optional)
- No, we decided to run the tests given on the cloud to get detailed performance of our service with the correctness ensured so that we won't waste too much budge both on improving performance and ensuring correctness.
- Describe an alternative design to your system that you wish you had time to try.
- Use multiple threads to compute PoW in blockchain.
- What were the toughest roadblocks that your team faced in Phase 2?
- Still ETL, it fails very often and still has some bugs even after we finished the design, costing a significant budget.
- Did you do something unique (any cool optimization/trick/hack) that you would like to share?
- Switching from HDinsight to Databricks is a life saver decision for us, we were able to better monitor and debug the ETL process.

Question 10: Contribution

- In the Phase 1 Final reports, we asked about how you divided the exploration, evaluation, design, development, testing, deployment, etc. components of your system and how you partitioned the work. Were there any changes in responsibilities in Phase 2? Please show us the changes you have made and each member's responsibilities.

In phase 2, zhitong primarily worked on etl and storage tier, yuwei primarily worked on web tier, kevin worked on all three tiers. So there is not much change in responsibilities, only that two of the members worked on storage tier in addition to the original responsibilities.

- Please show Github stats to reflect the contribution of each team member. Specifically, include screenshots for each of the links below.

  https://github.com/CloudComputingTeamProject/<repo>/graphs/contributors

  https://github.com/CloudComputingTeamProject/<repo>/network