Team Project Phase 2 Report

Summary

Team name: TheCloudCrew
Members: yujunl2, Jialiny3

Name	Andrew ID
Yujun Lee	yujunl2
Jolene Yu	Jialiny3

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

Live Test Performance & Configurations

Number and types of instances: 6, m5.large
Cost per hour of the entire system: 0.6648
(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: 12
Live test submission details:

	Blockchain	QR Code	Twitter	Mixed Test (Blockchain)	Mixed Test (QR Code)	Mixed Test (Twitter)
score	14.3	4.5	0	10	8	0
submission id	N/A	N/A	N/A	N/A	N/A	N/A
throughput	21364	20242	0	10225	15877	0
latency	N/A	N/A	N/A	N/A	N/A	N/A
correctness	100	100	0	100	100	0
error	0	0	100	0	0	100

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).
    1. Unified Service Cluster: Using 5 instances of m5.large with each node supporting all services. This setup provides resilience since all 15 service replicas are distributed across each node, ensuring each node can handle multiple services simultaneously, which improves availability if a node fails.
    2. Service-Specific Cluster: Using 7 instances of m6g, each dedicated to specific services. This separation can optimize resource utilization for complex, mixed queries by isolating workloads and reducing contention between services, though it may underutilize resources for individual services like blockchain, where only a subset of instances is needed.
- Discuss how your design choices mentioned above affect performance, and why.
  The m6g setup allows us to separate workloads, which can improve performance for mixed queries by reducing inter-service competition on each node. However, this approach may lead to inefficiencies for services that do not need all 7 instances, causing idle resources. The m5.large setup avoids this issue by distributing all replicas across each node, ensuring that all instances are fully utilized.
- 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 plan to switch to ARM-based m6g instances to test their performance across services. This choice allows more flexibility in dedicating instances to specific services, which may benefit complex queries and improve performance in certain scenarios. Initial tests indicate potential gains in efficiency and processing speed, although the effectiveness will depend on the specific workload distribution.

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?
  Phase 1:
  userInfo (usr_id PRIMARY KEY, screen_name VARCHAR(22), description LONGTEXT)
  interactHashtagScore (usr1, usr2 , hashtag_score, interaction_score, PRIMARY KEY (usr1, usr2), FOREIGN KEY (usr1, usr2) REFERENCES userInfo(usr_id)
  contactTweets (sender_id interacted_user_id , type, tweet_id BIGINT NOT NULL, created_at, text hashtags, PRIMARY KEY (sender_id, tweet_id), FOREIGN KEY (sender_id,interacted_user_id) REFERENCES userInfo(usr_id)

  For Phase 2, we added indexes to optimize performance by enabling faster lookups. The composite indexes on userInfo(usr_id), interactHashtagScore(usr1, usr2), and contactTweets(sender_id, interacted_user_id) reduce query time by quickly locating user pairs, which significantly speeds up searches for specific users or interactions. This adjustment improved query performance, especially when filtering by usr1 or usr2 in complex queries.
- Compare any two schemas for Twitter. Try to explain why one schema is more performant than another.
  Adding indexes in Phase 2 made this schema more performant than the Phase 1 schema because indexes reduce the need for full table scans, allowing for quicker access to specific rows. A composite index on user pairs, for example, enables efficient querying of interactions between users without the database needing to examine every record. This indexed structure is particularly beneficial for user-based lookups and speeds up frequently executed queries.
- 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?
  1. User Pair Standardization: By ensuring user1 always has a smaller ID than user2, we eliminate redundant pairs and reduce storage overhead, making lookups and comparisons faster.

2. Precomputed Scores Table: Storing precomputed hashtag and interaction scores minimizes real-time calculations, shifting this work to the ETL phase. This design reduces the load on the database during querying, speeding up response times.
3. Indexing: Adding indexes on frequently queried fields, as implemented in Phase 2, accelerates data retrieval by allowing the database to access specific rows directly rather than scanning the entire table. This approach is particularly useful for complex or repetitive 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
  We used Scala for the ETL process within the Zeppelin environment, which is efficient for data processing and offers strong compatibility with Spark, making it well-suited for large-scale ETL operations.
- The number and type of instances used during ETL and justification
  An HDInsight cluster was used for ETL, as it provides scalable resources for processing large datasets efficiently and integrates smoothly with Spark, minimizing configuration overhead and enhancing performance.
- The execution time for the ETL process, excluding database load time
  The ETL process took approximately 30 minutes.
- The time spent loading your data into the database
  Loading the ETL-processed data into the database also took around 30 minutes, allowing us to complete the overall process within an hour.
- The overall cost of the ETL process
  This was influenced by HDInsight usage and storage, though exact cost details would depend on the specific rates for compute and storage resources utilized.
- The number of incomplete ETL runs before your final run
  We saved four intermediate databases throughout the ETL runs, allowed us to preserve partial results and troubleshoot issues without restarting the entire ETL process.

- The size and format of the backup
  We backed up the resulting data in CSV format, which is efficient for large datasets and provides broad compatibility for future processing or restoration
- Discuss the difficulties encountered
  We encountered issues with delimiters and line separation, which affected data parsing. Adjusting the configuration to handle different formats helped us resolve these issues.
- If you had multiple database schema iterations, did you have to rerun your whole ETL pipeline between schema iterations?
  No, intermediate result is store so only part of the work need to be done between schema iteration. For example cleaned data is stored..

- 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 saved ETL output as a CSV in cloud storage and loaded it into the database using SQL's LOAD command. An alternative could be connecting Zeppelin directly to the database for real-time insertion, but this approach is costly and requires ETL re-runs for each new database creation.
- Did you store any intermediate ETL results and why? If so, what data did you store and where did you store the data?
  We saved intermediate ETL results after the data cleaning phase, avoiding repeated processing. This approach reduced ETL costs and improved efficiency by allowing us to reuse large, frequently accessed datasets.
- 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]
  Sorry but currently we are not able to get the ETL running and we have no extra time to do this
- 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.
  Before running Spark tasks in a cluster, we performed tests on a smaller dataset, validating filter rules and identifying edge cases. This approach ensured our ETL logic was sound before scaling to the full dataset. WE do not have screenshot of test coverage but I do create my own test case that run in local services and compare result to the reference mini-database services:
  http://localhost:8888/twitter?user_id=73774518&type=both&phrase=La&hashtag=familia

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.
  DB columns type: currently our ID has VARCHAR type which can take up many space, by converting them to number(INT) can be more efficient for comparaction.
  Creating index: We have not create any index currently, we are considering create composite index on userID and interacted ID to make the search more efficient.
- 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.
  Due to time constraints, we were not able to apply the tweaks and test the performance.

- For every configuration parameter or tweak, try to explain how it contributes to performance.
  DB Columns type: Integer types (INT) take up less space than strings in memory, reducing I/O and comparison times. Sorting, indexing, and joins involving integer keys are much faster than those involving strings, improving overall database query performance.
  Creating Index: Creating composite indexes would allow the database to quickly locate rows that match a query, rather than scanning the entire table. Composite indexes on common query column values such as userID and interacted ID can significantly speed up the execution, reducing load and improving response times.

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.
  1. Blockchain, instead of implementing another validation services, we incorporated the whole validation service into the creating new block services.
  2. CI/CD: We were creating the script in README.md in phase 1, currently we write bash script that can be run directly to create cluster and build image automatically, we are considering to build teh github action.

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 were currently using Azure Storage to store the csv, we will switch to using amazon S3 storage once all the database has been cleaned and ready to be processed.

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?
  Key CloudWatch metrics for monitoring system health include CPU utilization, memory usage, disk I/O, and network throughput. Tracking CPU utilization of each worker node, as used by our team, helps identify potential processing bottlenecks and overall cluster health. For performance insights, monitoring request latency, error rates, and the number of requests per second can reveal if services are slowing down or encountering issues.
- Which statistics did you collect when logged into the EC2 VM via SSH? Which tools did you use? What do the statistics tell you?
  When accessing the EC2 VM via SSH, tools such as top, htop, vmstat, and iostat can provide valuable statistics. Metrics like CPU usage, memory utilization, disk I/O rates, and active processes help gauge resource load and identify potential bottlenecks. These statistics offer insights into system performance, helping us diagnose resource exhaustion or process-level issues.
- Which statistics did you collect using kubectl? What do the statistics tell you?
  Using kubectl top nodes, we gather CPU and memory utilization metrics for each Kubernetes node, which helps identify potential load imbalances. This information reveals whether nodes are approaching capacity and need scaling or redistribution of workloads to maintain performance and stability.
- How did you monitor the status of your storage tier? What interesting facts did you observe?
  Monitoring the storage tier involves tracking metrics like disk usage, IOPS (input/output operations per second), and latency. Anomalies in these metrics, such as unexpectedly high latency or increased disk usage, could indicate storage bottlenecks. Observing trends here helps maintain optimal storage performance and avoid issues impacting data-intensive applications.
- 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.
  To maintain service continuity during failures, we could implement caching mechanisms

to serve limited requests even when back-end services are down. Additionally, using redundancy techniques, retry mechanisms, and load balancing across instances can ensure requests are still served, albeit with potential latency. This approach mitigates risks from crashes, network issues, or VM terminations, enabling graceful degradation.

- 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?
  During live testing, our QR Code Service experienced significantly low throughput, possibly due to inadequate warm-up or resource contention with other services. This unexpected behavior led to a significant performance drop, reducing our score by two-thirds. To address this, we might look into pre-warming instances or adjusting resource allocations, which could help stabilize performance under high load conditions.

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.)
  The design would work with 10x or even 100x more data, given a proportional increase in machines. Storing cleaned and intermediate results reduces redundant processing, and indexing minimizes the impact of larger data on query time. However, latency might increase due to the sheer data volume, but the indexing strategy should help keep query time manageable.
- Would your design work if your web service also implemented insert/update (PUT) requests? Why or why not?
  The current schema can handle insert/update requests because hashtag and interaction scores are precomputed. When new data is added, scores can be recalculated only for affected user pairs. However, this approach may become inefficient with frequent updates, as recalculating scores for each update can introduce significant overhead.
- 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?
  To efficiently handle insert/update requests, I would remove precomputed scores from the database. Instead, I'd maintain the userInfo table alongside a table for user pair interactions, storing tweet details, types, and the latest tweet information. This setup reduces the need for recalculation on each update, as scores could be computed dynamically or cached, enhancing performance for dynamic data while minimizing storage redundancy.

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?
  We set up a local development environment by installing a mini-database on my machine for experimentation. We packaged services and ran them locally to verify functionality before deploying them to the Kubernetes cluster. Each time we packaged services, we also tested them by running the local image in a container and ensuring connectivity.
- Did you attempt to generate a load to test your system on your own? If so, how? And why? (Optional)
  To simulate load, I wrote a script with a while loop that continuously sent requests to the system. Additionally, I created specific database queries to test connection stability and measure response times. This approach helped us identify performance bottlenecks and validate system resilience under high traffic.
- Describe an alternative design to your system that you wish you had time to try.
  An alternative design would involve more advanced caching mechanisms and sharding of the database by user ID, which could reduce latency and improve scalability. Sharding the data across multiple database nodes would allow us to handle larger datasets more efficiently, particularly if insert/update requests were frequent.
- What were the toughest roadblocks that your team faced in Phase 2?
  One of the toughest challenges in Phase 2 was the discrepancy between our live test results and the Sail Grade feedback we received. Despite thorough testing and optimizations, the live environment exhibited performance issues that were not visible during local testing, likely due to differences in workload and resource availability.
- Did you do something unique (any cool optimization/trick/hack) that you would like to share?
  One optimization we implemented was caching intermediate ETL results in cloud storage instead of re-running the ETL pipeline every time. This saved us significant processing time and reduced costs, as we could reuse large datasets rather than reprocess them for every test or data load. This approach optimized our workflow and made iterative development more efficient.

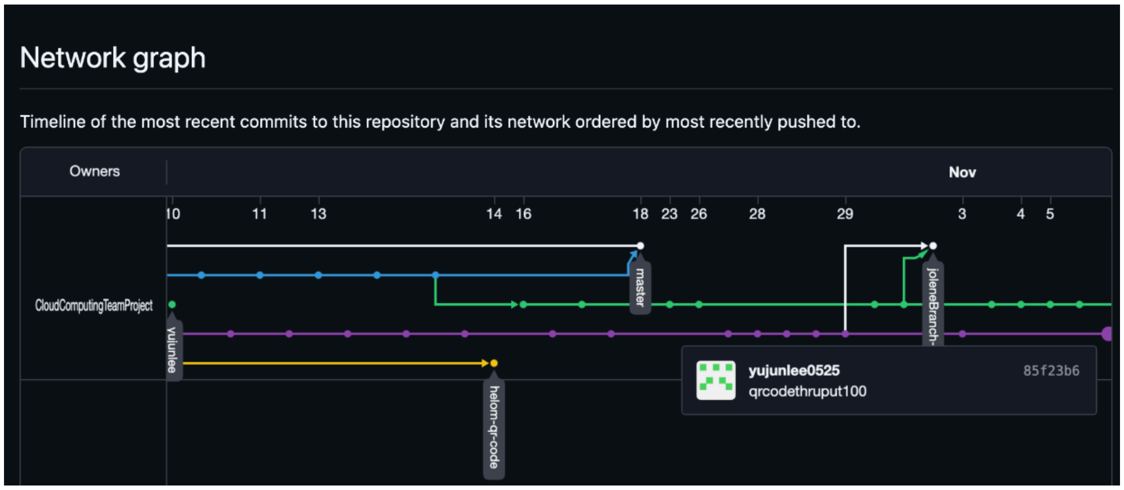
Question 10: Spark and ETL Process

- Did the concepts and tasks from Project 3 (Spark project) assist you in your ETL process for the Team Project? If so, in what ways did they contribute to your understanding or implementation? How did the Spark tools or techniques help, if at all, in optimizing or managing the ETL pipeline? Would you approach the ETL process differently without Spark?
  Yes, the concepts and tasks from Project 3 (Spark project) were invaluable for the ETL process in our Team Project. I was able to reuse much of the code and techniques to streamline ETL tasks, particularly in data cleaning and transformation. Project 3 provided insights on using partitioning and indexing to optimize performance, which directly

influenced our approach by allowing us to handle large datasets more efficiently. Spark's distributed processing model made it easier to scale our ETL pipeline, breaking down data into manageable partitions and reducing runtime. Without Spark, I would approach the ETL process differently, relying on sequential processing, which would be significantly slower and harder to scale, especially for large datasets.

Question 11: 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.
  Yes, Jialin deals with the blockchain services and Yujun worked witht QRcode. Instead of each team member is dealing with different component we now switch to each person responsible for a service. But we plan to work together for the ETL process as it is more complicated and need more effort.
- 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


Network graph
Timeline of the most recent commits to this repository and its network ordered by most recently pushed to.
Owners
Nov
10
11
13
14
16
18
23
26
28
29
3
4
5
CloudComputingTeamProject
yujunlee
master
joleneBranch-
helom-qr-code
yujunlee0525
qrcodethruput100
85f23b6