Team Project Report - Phase 3

Phase 3 summary

Team name: cc-team
Members (names and Andrew IDs):
Long Wang (longw), Xianting Wang(xiantinw), Ke Lu(klu2)

Performance data and configurations

Please note down the following information for your service in the live test.
Number and types of instances:
MySQL: 7*m5.large instances (each with 100GB EBS volume), one application ELB
Cost per hour of the entire system:
MySQL: $0.795 per hour
Your team's overall rank on the scoreboard: 35
Queries Per Second (QPS) for each query:

	Q1	Q2H	Q2M	Q3H	Q3M	Q4H	Q4M
score	11.00		15.00		15.00		3.21
submission id	275766		275767		275768		275769
throughput	28963.61		22867.40		2537.24		428.39
latency	3.00		2.00		19.00		62.00
correctness	100.00		99.00		98.00		47.00
error	0.03		0.05		0.91		0.19
rank							

Rubric

- Each unanswered bullet point = -4%
- Each unsatisfactory answer = -2%
- Use the report as a record of your progress, and then condense it before sharing with us.
- 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.

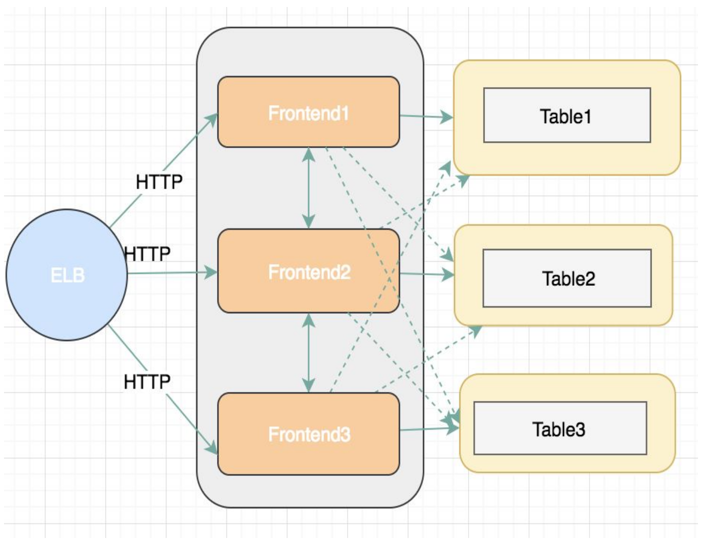
Task 1: Improvements of Query 1 & 2 & 3

Question 1: Web-tier architecture

You are allowed to use several types of EC2 instances, and you are free to choose which service each instance provides (e.g. use several dedicated front end servers + some DB instances; or every VM has a DB and a front end). When you try to improve the performance of your web service, it might be helpful to think of how you make use of the available resources within the provided budget.

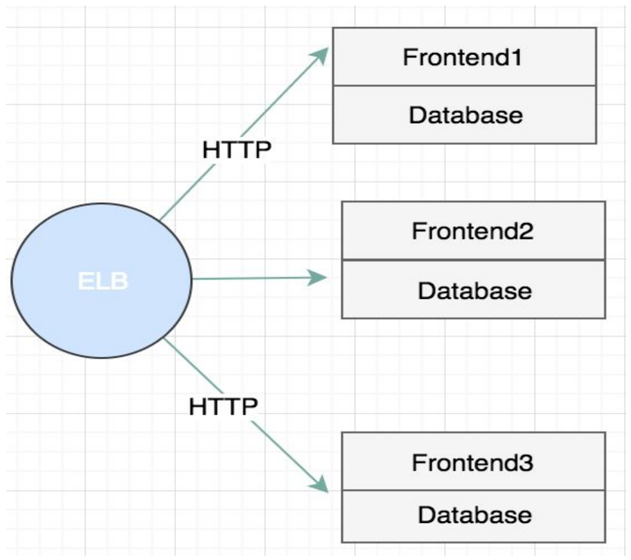
- List at least two reasonable varieties of web-tier architectures. (E.g. Use graphs to show which pieces are involved in serving a request.)

1. Separated instances and sharding database:



As is shown in the graph, front-end servers and database servers are separated in different instances, and the objective is to relieve the workload and pressure of each instance. Moreover, MySQL database is sharded according to tweet tid. HTTP requests arrives in load balancer and it routes to different front-ends, each of which takes care of handling a range of uuids. If the frontend receives its corresponding uuid, it will perform subsequent queries to the tables in another instance; otherwise it will redirect the HTTP requests to destined frontend.

2. Put identical Front-ends and database on each instance:



As is shown in the graph, front-end servers and database servers are put in the same instance. HTTP requests arrives in load balancer and it routes to different front-ends, each of which takes care of handling a range of uuids. If the frontend receives its corresponding uuid, it will perform database queries on local database; otherwise it will redirect the HTTP requests to destined front-end.

- For all instances in each type, indicate whether they are identical (for example, replicas), or if they serve different purposes (for example, shards of DB).
  - ❏ The first architecture discussed above is sharding, both for front-ends and database. Different instances serve for different purpose.
  - ❏ The second architecture discussed above is replica, each instance serves for same purpose except for that different frontend handles different ranges of uuids, but the deployment and frontend logics are identical.
- Discuss how your design choices will affect performance, and why. (Optional: show experimental results for all options.)
  - ❏ We have tried database sharding and replica. For sharding, since HTTP transfer is extremely slow, and the worst case ends up 4 runs of network communication for one frontend to finish a request before it sends back response. The throughput barely reached 100. For replica, since each frontend queries database locally, it saves the time to travel to another instance, our throughput has reached 2000-6000 after adopting replica.
- Describe your final choice of instances and architecture for your web service. If it is different from what you had in Phases 1 or 2, describe whether the performance improved and why.
  - ❏ As is discussed above, we chose replica of mysql and it is the same as Phase1 and Phase2. Our performance has improved for q2 and q3.

Question 2: Database and schema

If you want to make Query 3 faster, the first thing to look at is the DBs because the amount of computation at the front end might not be too heavy. As you have experienced in previous phases, various schemas can result in a very large difference in performance! Also, you might want to look at different metrics to understand the possible bottlenecks and think of what to optimize.

- What schema did you use for Query 3 in Phase 2? Did you change it in this phase? If so, how and why? How did the new schema affect performance?
  - ❏ We used MySQL replica and separated columns to store information including the basic data and keywords and impact score on one tweet base. We did not change it.
- Try to explain why one schema is faster than another.
  - ❏ We didn't change schema for query 3, but we did change schema for query 2. Previously, we had two tables and we queried database two times when handling each request. We reduced the query by using only one table. Front-end has to handle more but it is faster than query to database.
- Explain briefly the theory behind at least 3 performance optimization techniques (different from the ones in your Phase 2 report) for databases in general. Are they related to the problem you found above?
  - ❏ Configuration of database to appropriate engine types, cache size, max connection and etc. Different engine types accomodate to writing and read purpose.
- How are each of these optimizations implemented in either MySQL or HBase?
  - ❏ They are set through MySQL shell.

- Which DB did you choose and why?
  - ❑ We chose Mysql, because it has satisfactory performance in the implementation during all phases.

Question 3: 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 phases 1 and 2. To perform even better, you probably want to continue with your experimentation.

- List as many possible items to configure or tweak that might impact performance, in terms of: web framework, your program, DBs, DB connectors, OS …
  - ❑ Web framework: use both asynchronous and synchronous mechanism within our choice of framework(undertow), increase IO threads and worker number of framework
  - ❑ Program: HTTP redirect instead of waiting for response; sending response before query database for "write" operation; set timeout; In terms of range search, query statement include a list of keywords instead of creating new query statements for every keyword
  - ❑ DB connectors: use connection pool and limit the size
  - ❑ OS: utilize memory of server(e.g.increase buffer pool size)

- Apply one optimization at a time and show a chart with their respective RPS.

original RPS	**highest RPS**	**increase**	**Optimization**
Q2_one m5.large_600 seconds			
450	1258.35	808.35	Change from 2 tables to 1
1258.35	1928.77	670.42	Combine Query in Frontend
1928.77	2226.52	297.75	Change IO threads
2226.52	4688.1	2461.5	Connection Pool
Q3_one m5.large_600 seconds			
666.13	1211.85	545.72	Connection Pool
Q4_7 m5.large_600 seconds			
3105.39	3762	656.61	timeout from 5s to 10s
3762	4074.54	312.54	Adjust workers of front-end

- For every configuration parameter or tweak, try to explain how it contributes to performance.

- ❑ Increase the capability of front-end server to handle a large number of threads. Since there's requirements for order, it is likely that some requests with larger seq number arrive first but cannot be handled right away. Many workers are occupied and further result in a block. To improve the capability of front-end servers, we increased the IO thread and worker numbers of servers, so that a large number of threads will not team up for workers and block our server.
- ❑ Increase the connection to query database, allowing multi-threads performing queries to database at the same time. In this case, connection is reusable and the time for creating new connection is saved; furthermore, multiple threads can perform queries to database simultaneously which greatly improve the speed. To increase the threads to query database, we created a connection pool and limited the pool size to 1000. Each time a query tries to query database, it will check if there's idle connection in the pool before creating a new connection.

Task 2: Query 4

Question 4: Schema

Query 4 is like Query 2 and Query 3 for the read operations. The major difference is that Query 4 is a mixed read/write query, so you probably want to think what schema(s) will give good performance for your DB.

- What different schemas have you designed and explored? Which one did you choose in the end? Why?
  - ❑ We have tried both sharding and replica as discussed in the task1. We used replication in the end. Different from query2 and query3, we changed the table engine from MyISAM to INNODB.
- How did the different schemas impact performance? We expect you to show the different experiments you ran to justify your decisions.
  - ❑ With sharding, we can achieve 2000 throughput with only 2 shards, but with 7 shards, the throughput was really low (~10). With replication, we could achieve ~6000 throughput with 7 replica.
- Try to explain why one schema is faster than another.
  - ❑ Our sharding was based on tid range so that every read operation needs to fetch data from all the shards. It turned out our sharding was extremely slow when we use 7 databases, and we think the reason was probably due to low speed in frequent HTTP communications. For replication, since it was guaranteed that two uuid will not touch the same tid, so each frontend only need to connect to its local database, which saved a lot of time for communication.
- Which DB did you choose for Q4 and why?
  - ❑ We chose MySQL for Q4 because MySQL gave us good throughput on Q2 and Q3 and we want to keep consistent with our database.

Question 5: Consistency

Query 4 requires operations to be performed in a particular sequence specified in the request. Reads need to be blocking but writes can return immediately while performing the operation later.

- Explain your configurations of your front end system for Q4. Did you follow an approach similar to project 3.3?
  - ❑ Yes, we divided front-ends according to uuid. Each front-end is responsible for handling a range of uuids, so that the consistency is guaranteed.
- How did you control the execution order by the sequence number while achieving good performance at the same time?
  - ❑ We used a hashmap to keep track of the sequence number that is being executed for certain uuid, with key is uuid and value is the sequence number being executed. One thread was responsible for executing one request, and they need to wait until the previous sequence number was executed. And we set timeout to be 10 seconds, if one thread waited for 10 seconds, we dropped the thread.
- What have you done on your frontend to improve the write performance?
  - ❑ We sent response message before performing query to database; created new threads to allow multiple requests and responses at the same time.
- What are some other improvements you can think of but haven't tried?
  - ❑ It may be helpful to use a cache in the frontend to utilize memory instead of querying database too often.

Question 6: Profiling

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

- Given a typical request-response for **each of** queries 1-4, describe the latency of each of these components:
  - a Load Generator to Load Balancer (if any, else merge with b.)
  - b Load Balancer to Web Service
  - ❑ It measures the time elapsed in seconds after the request leaves the load balancer until the web service receives the request. Latency may result in Network connectivity, ELB configuration, backend instance issues(CPU, memory, etc).
  - c Parsing request
  - ❑ This latency refers to the time a request arrived in server and the time server parses it as a HTTP request.
  - d Web Service to DB
  - ❑ It refers to the when a HTTP request starts to be handled to the time it is sent to DB. If the number of IO threads is not sufficient, or the connection doesn't close, request is queued up and latency is increased.
  - ❑ For Q4, since each request needs to wait for the previous one to complete, so this step becomes the major latency.
  - e At DB (execution)

- ❑ For database, latency refers to the time from query starts to finishes. Latency can be high because locality is bad, cache is full, etc.

f DB to Web Service

- ❑ It refers to the time for database to send back response to web server. The time cost from DB to Web service is negligible. It is determined by database network latency (measured by ping).

g Parsing DB response

- ❑ This latency refers to the time a response from DB arrived in server and the time server parses it as a HTTP response.

h Web Service to LB

- ❑ It measures the time elapsed between response is received by web server and the load balancer receives the response.

i LB to LG

- ❑ It measures the time elapsed between response is received by load balancer and the load generator receives the response.

- For each step, think about how you can measure **each** latency data, and write down the tools / techniques that you used to do this.
  - ❑ The overall latency of load balancer could be retrieved from CloudWatch.
  - ❑ The total time elapsed (in seconds) from when the load balancer sends the request (HTTP listener) to a registered instance and the instance begins sending the response can be measured using ELB access log. Usually, with a throughput of about 1000-2000 RPS per instance, around 0.5ms is acceptable.
  - ❑ Also, we used instrumental profiling to record start time and end time, saved the time cost at each step(including time for each query at DB, DB to Web service) and printed them to the console.

Task 3: Development and Operations

Question 7: 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. Setting servers up automatically saves developers from repetitive manual labor and reduces errors. In this phase, you have more data that you have to load into the system, and perhaps you want to make sure you can start the clusters before the live test without any last-minute drama. It is worthwhile to devote some time into the auto-deployment of your service.

- How did you package and automate the deployment of your system? Describe the steps your team had to take every time you deployed your system.

- We launched instances, downloaded data from s3 bucket, installed mysql server and maven server, loaded data into MySQL and created index. The operations on MySQL are put in sql file and the other commands are in a bash file.

- What are some other ways to save a snapshot / backup of your server environment and databases?
  - A snapshot or an AMI can help to backup server environment and database, but due to "lazily loading", it was not reliable.
- (Optional) Can you automate the entire deployment process?
  - Yes, with the assistance of terraform and bash file.

Question 8: Live test and site reliability

Phase 3 is evaluated similarly to Phase 2 differently than how we evaluated Phase 1. In Phase 3, you can make as many attempts as you want by the deadline. However, all these attempts do not contribute to your score. Your team's Phase 3 score is determined by the live test as in Phase 2. In Phase 3, there is only one live test and your team decides which DB to deploy. The live test 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; in case 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 understand performance?
  - Request count of ELB and CPU utilization of each EC2 instance are helpful to monitor the health.
- What statistics can you collect when logged into the VM? What tools do you use? What do the statistics tell you?
  - We can get the CPU utilization, memory usage and disk usage when we log into the VM. We used simple command line tools like "top", "vmsta", "htop"to get those information on a normal ubuntu VM. We found that the disk usage for a table almost triples the original data size, and MySQL data loading never uses more than one core (CPU utilization always below 100%). When the system is running healthily, the CPU utilization is about 30% for front-end server and MySQL used up about 20%, and the RAM is used at its ⅛ capacity.

- How do you monitor the status of your database? What interesting facts did you observe?
  - We use command line 'MyTop' to monitor MySQL. We found that before we created a connection pool in front-end server, there's only one thread that's querying database at one time; but after we had a connection pool, multiple threads can access database at the same time.
- Should something crash in your system, was (and how) your service still be able to serve requests (although perhaps slower)? Think about this in different scenarios, such as front end, back end, or even an entire VM failure.
  - For front end crash, since each frontend is responsible for handling certain uuids, we will lose that portion of requests (1/7 for each instance).

- ❑ For MySQL database crash, also because each frontend is responsible for handling certain uuids, and each frontend will connection to its local database replication, so we will lose that portion of requests as well.
- ❑ For entire VM failure, the same thing will happen.

- 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?
  - ❑ Yes. Our throughput dropped a lot at around 8:50 pm during Q3 live-test. The impact was huge because it continued on Q4 and mixed tests. We could achieve around 6000 throughput for Q4 in 600s submissions, but almost got nothing during live-test for Q4 and mixed query. We tried to restart frontends and databases, reset database cache but nothing worked. We guess the reason might be something with the EC2 instance IO but we are not sure, since we never experienced this before even with tons of submissions. We ended up with poor performance for the live-test.

Task 4: Reflection

Question 9: Scalability

In this phase, we serve read and write requests from tens of GBs of processed data. Remember this is just an infinitesimal slice of all the user-generated contents at Twitter. Can your servers provide the same quality of service when the amount of data grows?

- 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.)
  - ❑ It probably won't work as well as it does now. Because we used database replica, the volume of the database is increasing and it may lead to unsatisfactory speed and heavy workload on memory. However, the negative influence is mitigated by a proportional increase of machine numbers.

Question 10: 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 the cloud VMs?
  - ❑ For ETL, we tested every E & T step locally before moving to the whole dataset. We also installed MySQL locally tested loading data into MySQL, but not with HBase.
- Did you attempt to generate a load to test your system on your own? If yes, how? And why?
  - ❑ No, we did not generate a load to test our system. We used submission for testing correctness and throughput.
- Describe an alternative design to your system that you wish you had time to try.
  - ❑ Use a hybrid database -- Hbase for Q4 and MySQL for Q2 and Q3.

- Which was/were the toughest roadblocks that your team faced in Phase 3 especially for query 4?
    - ❏ Throughput is not stable during livetest. The fluctuation is caused either by too many threads and lack of workers, or the fact that IO credit on our instance is running out.
- Did you do something unique (any cool optimization/trick/hack) that you would like to share?
    - ❏ No, we did not do have anything unique.

Task 5: Summary

After Phase 3, you are ready to make important decisions and choices about SQL vs NoSQL, trade-offs between different Cloud Service Providers, sharding vs. replication, MapReduce vs Spark and go out into the industry.

- Which database did you use for Phase 3? Explain why and compare the advantages and disadvantages of MySQL and HBase for each query.
    - ❏ We used MySQL for Phase3.
    - ❏ For Q2 and Q3, since we applied replication with MySQL, each frontend only need to communicate with its local database, which saved time on internet IO. But HBase is more scalable if we have larger datasets.
    - ❏ For Q4, since we are guaranteed that two uuids will not touch the same tid, MySQL is applicable in this situation and each frontend need to write to the local database. But if we want to keep the strict consistency, each write operation needs to be sent to all the databases and we need complicated locks for doing this. HBase is more straightforward to use when dealing with write operations.
- Which Cloud Service Provider did you use for ETL (In Phases 1, 2, and 3)? Why?
    - ❏ We used GCP for ETL because ETL was time-consuming and we want to save budgets on AWS.
- Which programming framework did you use for ETL? (eg. Mapreduce, Spark, other)
    - ❏ We used Mapreduce for ETL.
- What is the coolest optimization/hack/learning you have achieved throughout the team project? (eg. Scalability, Performance, etc)
    - ❏ We created a naive connection pool for MySQL database connection and it improved our performance a lot (almost doubled the throughput).