Team Project Report - Phase 2

Phase 2 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: 1*m4.large + 5*m5.large instances (each with 200GB EBS volume), one application ELB

Hbase: 6*m4.large instances as EMR (1 master + 5 cores, each with 200GB EBS volume), one application ELB

Cost per hour of entire system:

MySQL: $0.792 per hour

HBase: $0.792 per hour

Your team's overall rank on the scoreboard:

28

Queries Per Second (QPS) of queries:

	Q1	Q2H	Q2M	Q3H	Q3M
score	6.00	3.90	4.06	3.92	10.00
submission id	259434	259435	259695	259437	259696
throughput	30134.18	3117.40	3250.87	587.75	2271.71
latency	3.00	15.00	15.00	83.00	21.00
correctness	99.00	100.00	100.00	99.00	100.00
error	0.03	0.00	0.05	0.07	0.07
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.

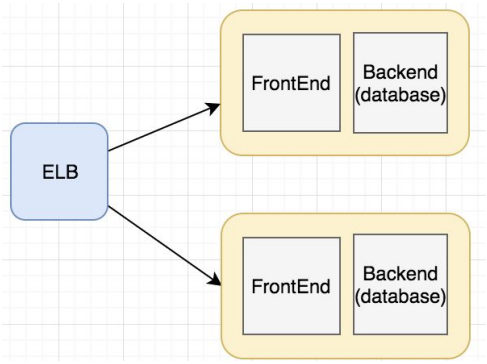
Task 1: Improvements of Query 1 & 2

Question 1: Web-tier architecture

You are allowed to use several types of EC2 instances, and you are free to choose what 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.

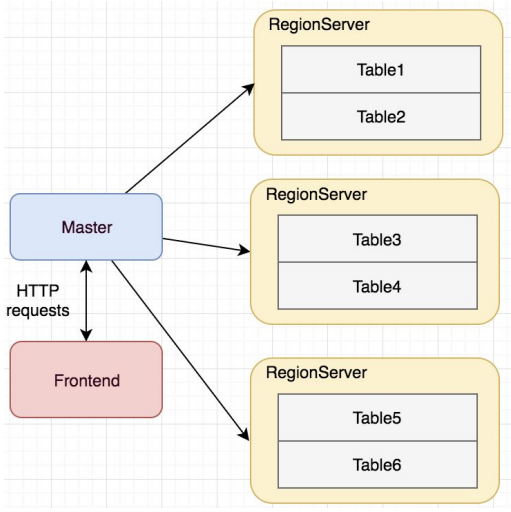
- List at least two reasonable varieties of web-tier architectures. (E.g. Use graphs to show which pieces are involved in serving a request.)

Every VM has a whole database and a frontend, known as replica.



- ❏ As is shown, frontend and database are put in the same instance, and the deployment is same for all instances. ELB is responsible for receiving http requests and distribute them among instances. The frontend on each instance then handles the requests, parse the requests, sends queries (if necessary) to the database running on the same instance, and finally writes response of each request to ELB.

Every VM has partial database , known as sharding.



- ❏ The database is deployed on a EMR cluster, and used a separate instance as frontend server. The frontend server is responsible for receiving http requests, parsing the requests and sending queries (if necessary) to the database through connection. After receiving the query results, the frontend will organize results and write responses .

- 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).
  - ❏ For the first type which we used for Mysql, all instances are identical. For the second type which we used for Hbase, each instance holds parts of databases and they serve for different queries.
- Discuss how your design choices will affect performance, and why. (Optional: show experimental results for all options.)
  - ❏ For mysql, we choose replicas and frontend is deployed in each instance. The advantage is that: the connection time between frontend and backend can be decreased; also, if one instance is down, other instances will not be influenced.
  - ❏ For hbase, we choose sharding. The advantage is that each backend is responsible for a smaller dataset so that the performance can be improved.
- Describe your final choice of instances and architecture for your web service. If it is different from what you had in Phase 1, describe whether the performance improve and why.
  - ❏ Major change is made for HBase. We had one dedicated instance to hold frontend in Phase 1 and changed to 6 frontends which were deployed in master nodes and core nodes respectively.
  - ❏ We didn't change for MySQL.
  - ❏ As a result of both choices, the performance didn't improve, because our bottleneck lies in schemas instead of web services architecture.

Question 2: Database and schema

If you want to make Query 2 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 Phase 1, various schemas can result in 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 Query 2 in Phase 1? Did you change it in this phase? If so, how and why? How did the new schema affect performance?
  - ❏ We used two tables for Query 2. In the first table we aggregated hashtags based on unique (keyword, uid) pair, and in the second table we aggregated hashtags based on unique keyword. When the requests come in, we then query the first table with matched (keyword, uid) pair and query the second table with matched (keyword), and combine these two records to get the final result. We did not change the schema in this phase.

- Try to explain why one schema is faster than another.
  - ❏ One schema is better than another if the same query can be done faster. Especially for hbase, if Get can be used instead of Scan, then such schema would be better for improving query performance.

- Explain briefly the theory behind at least 3 performance optimization techniques (different from the ones in your Phase 1 report) for databases in general. Are they related to the problem you found above?
  - ❏ Build index. Indexing on field means we have records sorted based such field, so when we want to retrieve one specific value, binary search could be used to decrease time complexity to O(log n).
  - ❏ Configure connection pool size. We could increase the connection size to allow more concurrent connections (like read operations) to the same database.
  - ❏ Enable database caching in memory. Databases could store recent query results in memory in order to speed up future queries.

- How are each of these optimizations implemented in MySQL and in HBase? 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)?
  - ❏ Build index: Hbase supports 'CREATE INDEX ON (field)' operation to build index. Hbase only uses row_key as their primary index. Secondary index could be implemented using another table.
  - ❏ Connection pool: By modify configuration files for both MySQL and Hbase, we could change the default connection size.
  - ❏ Memory caching: for MySQL we could set variables like query_cache_type and query_cache_size to enable and change cache size. For HBase we could set in_memory = true to allow caching in memory.

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 Phase 1. 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: we increased IO threads and workers in Undertow to improve performance.
  - ❏ Program: use priority queue to reduce the storage of useless data
  - ❏ DBs: In terms of configuration, we changed configuration for both databases. For mysql, we tuned the parameters including max-connections, backlog, concurrency, engine type to improve performance. For Hbase, we tuned the parameters including blocksize, compression, in-memory storage, and we used UI monitor to split server regions.
  - ❏ DB connector: we wrote connection pool to release connection after one query is fulfilled.

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

Mysql:

original RPS	highest RPS	increase	Optimization
881.08	1290.0	408.9	8_IO_threads_64 workers in Undertow
461.4	881.08	419.7	3000_max_connection
21	461.4	461.4	innoDB to MyISAM

Hbase:

original RPS	highest RPS	increase	Optimization
399.88	843.72	443.84	hbase config: in-memory
843.72	1572.13	728.41	hbase config: blocksize_4096
1572.13	3970.52	2398.39	hbase config: split region
3970.52	5572.8	1602.28	hbase config: snappy

- For every configuration parameter or tweak, try to explain how it contributes to performance.
  - ❏ Adjust max_connections in database and jdbc connection pool. The default connection size is not enough for large http requests and large concurrent

connections. By modifying configuration files for both MySQL and Hbase, we could change the default connection size.
- ❏ In-memory. Databases could store recent query results in memory in order to speed up future queries.
- ❏ Change innoDB to MYISAM engine type in Mysql. MYISAM implements the FULLTEXT index which is quite useful for integrating search capabilities.

Task 2: Query 3

Question 4: Schema

Query 3 is like Query 2 in many aspects: you can possibly use a similar front end and back end configuration, as well as the code that you developed to set things up and connect the different components. The major difference is that Query 3 is a range query, so you probably want to think what schema(s) will give good performance for both DBs.

- What different schemas have you designed and explored? Which one did you choose in the end? Why?
  - ❏ We firstly used to the schemas 1 (SELECT * FROM q3_tweet WHERE uid < ? AND uid > ? AND time < ? AND time > ?). The execution time is very long. Then we changed to schemas 2 (SELECT text, freq, score, tid FROM q3_tweet FORCE INDEX (uid_time) WHERE (uid BETWEEN ? AND ?) AND (time BETWEEN ? AND ?)"). We also tried to use JOIN to implement the range query.

- How did the different schemas impact performance? We expect you to show the different experiments you ran to justify your decisions.
  - ❏ JOIN runs much slower than BETWEEN … AND … . We printed its execution time for each request and draw that conclusion. We also used "SHOW STATUS LIKE 'Handler_read%' to check the detailed information for each request, including handler_read_first, handler_read_key, handler_read_next. Then we realized that JOIN operation required more handler_read_next than BETWEEN… AND… operation.
  - ❏ Add INDEX uid_time (uid, time) is much faster than without adding it when we were creating a table for Query 3.

- Try to explain why one schema is faster than another.
  - ❏ The reason that JOIN runs slower than BETWEEN … AND … is that the database reads all ordered items and then filter out the ones that are not matched.

Question 5: 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 Q3, 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.
- ❏ After we scaled up the number of IO threads and worker threads, the time from web service to DB is negligible.

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.
- ❏ For Query 2, the time cost for each query after warming up is around a few ms; for Query 3, it is around 10+ ms.

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 latency, 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. Usually, with a throughput of about 1000-2000 rps, the time for each query is 1-3 ms for q2 and 10+ ms for q3.

Task 3: Development and Operations

Question 6: 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 has your automation changed in Phase 2.

- Did you improve your automated deployment process? What were the improvements or changes you made?
  - ❏ No, we did not improve our automated deployment process. We still used bash scripts and terraform for frontend deployment.
- In this phase, you have more data to deal with. Did you change your ways of backing up your databases? If you did, what are the changes?
  - ❏ No, we did not change the ways of backing up databases.
- After finishing Phase 1 and Phase 2, you might have realized that you need different approaches to save a backup/snapshot of your MySQL and HBase databases. How did you back up your HBase data?
  - ❏ We created one snapshot for each table of HBase, and exported snapshots to s3 buckets. To restore Hbase data, we simply first restored the snapshot from s3 bucket, and then restore table from snapshot.

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 by 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; 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?
  - ❏ For ELB, HTTP requests could be used for monitoring. For each instance, CPU utilization, network in/out could be used for monitoring. HTTP requests of ELB could help understand performance.
- 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', 'vmstat' to get those information on a normal ubuntu VM, and 'hadoop dfsadmin -report' to get information on EMR cluster. 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%).

- How do you monitor the status of each database? What interesting facts did you observe?
  - ❏ We use command line 'MyTop' to monitor MySQL, and use HBase web UI to monitor HBase. We found that total requests to one region is not always proportional to its data size, which requires manually splitting or merging data regions for HBase.
- 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 we used ELB to connect to six front-ends, as long as our ELB does not crash and at least one front end is still working, our service will still be able to serve requests.
  - ❏ For MySQL database crash, since we have six replications of all the data, our service will also still be able to serve requests. For HBase, since HDFS also has replications, so one node crash will be tolerated.
  - ❏ For entire VM failure, still since we use 6 identical machines for MySQL, our service will still be functional. But HBase suffers from single-point-failure, which means if the master node crashes, the entire system would crash and our service will not be responsive any more.
- 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?
  - ❏ No, we did not experience anything unexpected during live test.

Task 4: 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 contents at 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 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.)
  - ❏ HBase backend should be scalable because it's based on HDFS which distributes data among a cluster, but our MySQL backend might also work as well for a much larger dataset since we applied replication for database, and if we are give a proportional amount of machines, the overall throughput might still remain the same.

- Would your design work if your web service also implemented insert/update (PUT) requests? Why or why not?
  - ❏ Probably won't work for the MySQL database because we used MyISAM engine here. Because MyISAM tables supports table level locking and a write lock is employed

when the database is updated. And also since we are using replica with MySQL, our design may not meet the requirement of fast writing.

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 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.
  - ❏ We did not reach target throughput in this phase and we discussed about the potential issues with our system. We would like to try different schemas for Q2 and Q3 if we have time (probably in next phase). The basic idea is to reduce the total rows for each table for saving time on database query, and let front-end do more complicated work like parsing the information since our database query was really slow.
- Which was/were the toughest roadblocks that your team faced in Phase 2?
  - ❏ The toughest roadblock is to reach target throughput. For both databases, we tried different optimizations such as tuning configurations of databases and front ends, but could not reach the target throughput without a cache. We thought our major problem might still be the schema and we will spend some time optimizing our schema first during next phase.
- Did you do something unique (any cool optimization/trick/hack) that you would like to share?
  - ❏ No, we did not do have anything unique.

Question 10: Contribution

- In Checkpoint 1 Report and Phase 1 Final Report, we asked about how you divided the exploration, evaluation, design, development, testing, deployment, etc. components for 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, Xianting was responsible for preprocessing data (filtering and extracting), loading data into HBase, and tuning configurations for HBase. Long was responsible for loading data into MySQL and tuning configurations for MySQL. Lu was responsible for implementing frontends and optimizing performance for both databases.

- There is a tab showing your Github contribution stats. Just like the Phase 1 Final Report, please include a screenshot of it and reflect on the contribution of each team member.