Team Project Report - Phase 1

Phase 1 summary

Team name:cc-team

Members (names and Andrew IDs):

Long Wang (longw), Xianting Wang(xiantinw), Ke Lu(klu2)

Performance data and configurations

Number and types of instances:

Q1: 6*m4.large instances (each with 30GB EBS volume), one application ELB

Q2-MySQL: 3*m4.large + 3*m5.large instances (each with 200GB EBS volume), one application ELB

Q2-Hbase: 5*m4.large instances as EMR (each with 50GB EBS volume), 1*m4.large instance as frontend (with 30GB EBS volume)

Cost per hour of entire system:

Q1: $0.651 per hour

Q2-MySQL: $0.792 per hour

Q2-Hbase: $0.639 per hour

Queries Per Second (QPS) of your best submission:

	Q1	Q2HBase	Q2MySQL
score	20.00	25.00	25.00
submission id	231802	239527	240796
throughput	28023.30	6959.40	7109.00
latency	3.00	6.00	6.00
correctness	98.00	100.00	100.00
error	0.06	0.00	0.00

Rubric

- Each unanswered bullet point = -4%
- Each unsatisfactory answer = -2%
- Use the report as a record of your progress, and then condense it before submitting it.
- If you write down an observation, we also expect you to try and explain it.
- Questions ending with “Why?” need evidence (not just logic).
- Always use your own words (paraphrase); we will check for plagiarism.

Task 1: Web tier

Question 1: Comparing web frameworks

Query 1 does not involve database access and is a good opportunity for you to compare and choose a favorite web framework for future queries. Please try a couple of different web frameworks, achieve an RPS of 25k with one framework, and 20k with another, and answer the questions below.

- What frameworks have you tried? How did each of them perform? Please include numbers.
  - ❏ As is shown, we tested Vert.x and Undertow for q1. Undertow has better performance in throughput, while vert.x has better performance in correctness and error. But overall, their performance is close to each other. The comparison is listed as follows:

Submission id	Framework	Throughput	Correctness	Error	Latency	Effective throughput
231804	Vert.x	24090.70	100.00	0.00	3.00	24090.70
231802	Undertow	28023.30	98.00	0.06	3.00	27446.36

- For the two frameworks with the highest RPS, please list and compare their functionality, configuration, and deployment.
  - ❏ **Functionality**:

    Vertx and Undertow have many similarities regarding to functionality. Both of them support HTTP, TCP, blocking API and non-blocking API, asynchronous network framework. Also, both of them are lightweight and embeddable. Moreover, in the TechEmpower Benchmark, their performances ranked among top 10 with test data types including plaintext, json, fortunes, database-access. [1]

    However, generally speaking, Vertx supports more functionalities and languages than Undertow.

    Vertx's network is based on Netty, one of the most prevalent application of asynchronous event-driven network application framework NIO. Undertow's network is based on XNIO, which simplifies low-level I/O layer of NIO framework, according to official site of XNIO.[2]

    Vertx has its unique features. To start with, it uses an event bus for distributed system, in which the applications are a set of "verticles" communicating via an unstructured JSON distributed bus. And Vertx uses a non-blocking event loop to handle http requests. Also, Vertx supports more functionalities such as UDP, DNS, TCP, File System, Event Bus, Sockjs. And it supports multiple languages besides Java.

[1] "Web Framework Benchmarks - TechEmpower." https://www.techempower.com/benchmarks/. Accessed 17 Mar. 2018.

[2] "XNIO - JBoss Community." http://xnio.jboss.org/. Accessed 17 Mar. 2018.

- **Configuration**:
  Since both of them are lightweight, the configurations are simple. For Undertow, packages to establish simple application include undertow.core and undertow.servlet. For Vertx, vertx.core and vertx.web are required as dependencies to develop simple web application.
- **Deployment**:
  To deploy servlets in Undertow, you should create a DeploymentInfo, which includes servlets' information. Most of data inside DeploymentInfor matches data in web.xml. Then the servlets can be deployed in respective servlet containers. After deployment, you can call start( ) on DeploymentManager and write HttpHandler to handle incoming requests.
  To deploy in Vertx, you need to create mainVerticle. Verticle is a fundamental component which is similar to servlet, and it needs deployVerticle for deployment. However, since the deployment process is asynchronous and multi-threaded, extra work needs to be done in certain java environment, such as using JDBC API . [3]

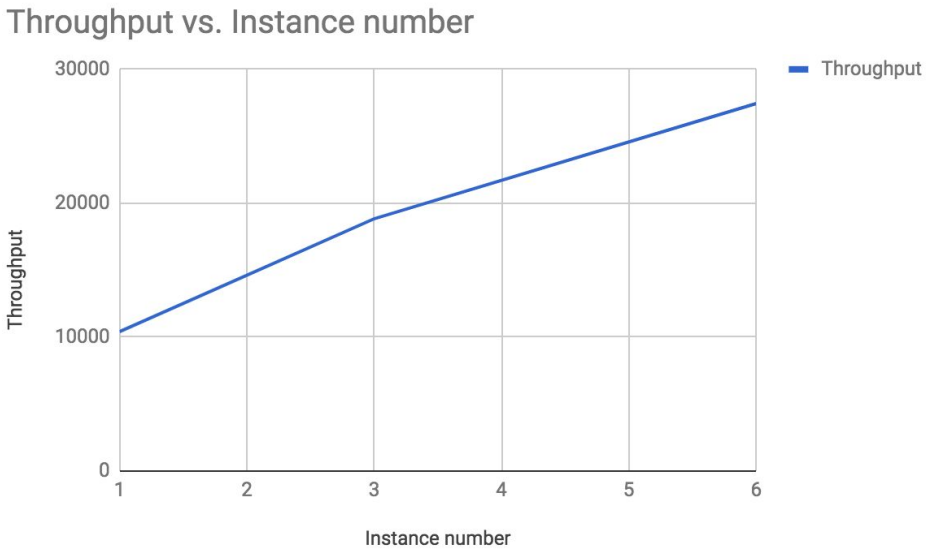
- Which one would you like to choose for other queries? What are the most important differentiating factors?
  - We are going to use Undertow for the following reasons. Firstly, Vertx discourages blocking API such as JDBC. Instead, Undertow is based on XNIO which provides API to combine blocking and non-blocking operations. To reduce redundant codes when connecting to thread pool, we are going to use Undertow instead. Secondly, from the test results of q1, the throughput of Undertow is higher than Vertx. Although the correctness is lower, it is tolerable. Thirdly, our team has more experience with servlets deployment rather than event bus in Vertx.

Question 2: Load balancing

Our target RPS for Query 1 is 25k, which can be very hard to reach with a single instance. We have learned about ELBs in the individual projects, so let us explore using it here. Please set up several instances with the framework you have chosen, adding one of them to the ELB at a time, and observe how the RPS changes.

- As the number of servers increases, does the RPS grow accordingly? Show a graph.
  - As is shown below, RPS grows when the number of servers increase.s, but not linearly.

[3] "Vert.x JDBC client - Vert.x." 9 Feb. 2018, http://vertx.io/docs/vertx-jdbc-client/java/. Accessed 17 Mar. 2018.



- How do you explain what you see in the previous question? Hint: draw a latency - RPS plot.
  - ❏ As is shown from the table below, with more frontend servers, the latency decreases and throughput increases. But the throughput does not increase linearly corresponding to the number of instances, because ELB needs to do some extra work when distributing the requests to registered targets, which introduced a overhead.

Instance number	Throughput	Latency
1	10405.56	8
3	18839.26	4
6	27446.36	3

- We mentioned ELB warm-up in individual projects. What is that and why is it necessary? What performance change do you observe during warm-up?
  - ❏ ELB warm-up refers to the process of running ELB to process requests for certain amount of time before official run.
  - ❏ AWS ELB runs on EC2 instance. ELB scales up or down based on traffic pattern, which is determined by AWS proprietary algorithms. When the instance number adds up and traffic pattern doesn't match the capability of ELB instance, users will experience latency. Thus, ELB can be configured properly to match required traffic pattern during warming-up, and to reduce latency in official running.[4]
  - ❏ Latency decreases and throughput increases during warm-up.

[4] "Top 5 ways to improve your AWS EC2 performance - Datadog." 6 May. 2016, https://www.datadoghq.com/blog/top-5-ways-to-improve-your-aws-ec2-performance/. Accessed 17 Mar. 2018.

- Discuss load-balancing in general, why it matters in the cloud, and how it is applicable to your web tier.
  - ❏ Load balancing refers to distribution of workloads among different resources. Commonly load-balanced systems include popular web sites, large Internet Relay Chat networks, high-bandwidth File Transfer Protocol sites, Network News Transfer Protocol(NNTP) servers, Domain Name System (DNS) servers, and databases.[5]Many load-balancing methods or scheduling algorithms are used. Random choice and round robin counts as simple algorithms. Sophisticated considerations make consider many factors to decide which server to send request to by a load balancer.
  - ❏ Cloud load balancing has an advantage over DNS load balancing as it can transfer loads to servers globally as opposed to distributing it across local servers.[6]Cloud computing brings advantages in "cost, flexibility and availability of service users." Thus, cloud load balancing serves to increase availability and utilization of resources.
  - ❏ The web tier has three functions: receiving incoming HTTP requests, caching and processing messages, sending HTML/XML back to client. A load balancer will be used to receive HTTP requests, and distribute the requests to different servers.

- (Optional) You can also try other software LBs, or even write your own LB. What are the benefits of each one and why would your own LB possibly outperform ELB? Hint: this might become clear in other queries.
  - ❏ We used AWS ELB in phase 1.

Question 3: Web-tier architecture

This question applies to all queries. We will need backend components that store and retrieve data from databases, as well as a frontend that interprets requests, fetches data, and does processing to generate responses. How would you set up servers for these purposes? You can use the following questions as a guide.

Keep in mind that, although you can test different queries individually for now, in later phases your servers will need to handle all queries at the same time.

- What web-tier architecture did you choose? Did you put frontend and backend on the same instance? (E.g. Use a graph to show which pieces are involved in serving a request.)
  - ❏ For mysql, we put one frontend and one database on each instance, and used one application ELB connecting to 6 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.

---

[5] "Load balancing (computing) - Wikipedia." https://en.wikipedia.org/wiki/Load_balancing_(computing). Accessed 17 Mar. 2018.

[6] "In demand: the culture of online service provision - Citrix." https://www.citrix.com/content/dam/citrix/en_us/documents/oth/in-demand-the-culture-of-online-service-provision.pdf. Accessed 17 Mar. 2018.

- ❏ For HBase, we deployed database 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 .

- What alternative architectures can you think of? What are their implications on performance and why?
  - ❏ For mysql, we could run frontends and backends on different instanced and apply sharding on the database to improve query performance. To be specific, we could divide the whole dataset into several shards according to primary keys and load each shard on one backend instance. The frontend servers are responsible for receiving requests and sending queries to each database according to the query keyword. Sharding will improve the query performance because each backend is responsible for a smaller dataset, but since we have limited budget, the number of total instances we could use for frontends and backends are reduced, which might harm our overall throughput.

- Explain your choice of instance type and numbers for your web tier.
  - ❏ We used m4.large and m5.large for all the queries in phase 1 because of requirement and budget limit.
  - ❏ For Q1: we used 6 m4.large instances and one application ELB.
  - ❏ For Q2-MySQL: we used 3 m4.large and 3 m5.large instances, along with one application ELB.
  - ❏ For Q2-Hbase: we ued 5 m4.large instances for EMR cluster(1 master and 4 cores), and one m4.large instance as frontend.
- (Optional) Have you tried more than one architecture or instance type / number? What did their performance look like? Can you reason about the difference?
  - ❏ Yes. For Q2-Hbase, we tested different number of cluster cores to compare the performance. The overall throughput improves when the number of cores increases, but not linearly. The difference is mainly because HBase could handle more queries concurrently with more nodes and thus improve performance.

Question 4: 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.

- How did you package and automate the deployment of your system? What steps were taken every time you deployed your system? Please include an auto-deployment script that sets up environment and starts the Query 1 server from a clean Linux VM under the **q1-vm-deployment** folder in your zip file.

- ❏ We set up target group, ELB instance and server instances with terraform, and run script to tag the resources. After instance starts running, we transfer project files and deployment script to install and run maven projects.

- What are some other ways to save a snapshot / backup of your server environment and databases?
  - ❏ Ways we have tried for backup: AMI, volume snapshot, s3 bucket.

- How did you automate the entire deployment process? Explain your solution here and include your scripts or Terraform config files under the **orchestration** folder in your zip file.
  - ❏ Firstly, we created a terraform script to deploy AWS resources, such as instance, elastic load balancer, target group, etc. Secondly, we save all command in a document and run the document on every instance.

Task 2: Extract Transform Load (ETL)

Question 5: Your ETL process

We have 1TB of raw data, which takes a non-trivial amount of time and money to process. It is strongly advised to test run and debug your E & T on a small portion of the data locally, and plan wisely before booting expensive clusters! For Q2, please write about:

- The programming model used for the ETL job and justification.
  - ❏ Step 1 - filter malformed data: streaming processing
  - ❏ Step 2 - transform data and aggregate hashtags: Hadoop MapReduce
- The number and type of instances used and justification.
  - ❏ Step 1 - filter malformed data: 1 instance on GCP, 2 cores and 8 GB memory
  - ❏ Step 2 - transform data and aggregate hashtags: EMR on AWS, 1 master and 5 core nodes, m4.xlarge
  - ❏ Step 3 - load data into MySQL: 3*m4.large and 3*m5.large (we directly load data into each MySQL server)
  - ❏ Step 4 - load data into HBase: EMR on AWS, 1 master and 2 core nodes, m4.large
- The execution time for the entire ETL process.
  - ❏ Step 1 - filter malformed data: ~13 hours
  - ❏ Step 2 - transform data and aggregate hashtags: 2 tasks, around 10 mins for each
  - ❏ Step 3 - load data into MySQL: 2.5 hours for two tables
  - ❏ Step 4 - load data into HBase: 2.5 hours for two tables
- The overall cost of the ETL process.
  - ❏ Step 1 - filter malformed data: ~$0.6
  - ❏ Step 2 - format data and aggregate hashtags: ~$0.5 (spot price)
  - ❏ Step 3 - load data into MySQL: ~$0.25 for each instance
  - ❏ Step 4 - load data into HBase: ~$0.5 (spot price)

- The number of incomplete ETL runs before your final run.
  - ❏ For step 2 MapReduce task, we failed twice because of a small bug in our code. For step 3 loading data into MySQL, we failed three times due to the decoding of data. We went smoothly with step 1 preprocessing data and step 4 loading data into HBase.
- The size of the resulting database and reasoning.
  - ❏ We used two tables for Q2, and the data files we loaded into databases are 2.5GB and 11.5GB in size. After loading, MySQL database used around 60GB of disk storage, and HBase used around 100GB of HDFS storage.
- The size and format of the backup.
  - ❏ For HBase, we used S3 for backuping database snapshot. The snapshots for two tables are 25GB and 5GB in size, respectively.
  - ❏ We tried to backup MySQL database to S3 but did not succeed in restoring the database. The backup was around 5GB in an zip file.
- Discuss difficulties encountered.
  - ❏ We had problems with restoring data from backup or AMI to MySQL database. Actually we was able to create new instances with our AMI and databases seemed to work when we tested with small number of requests, but when we submitted onto TPZ, it reported 'HTTP socket read timeout' exceptions.

Question 6: ETL considerations

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

- Which cloud platform did you use and what was the difference?
  - ❏ We used GCP to preprocess data (filtering) and used AWS EMR for further transforming data. The filtering step was quite time-consuming and required no parallel processing, so we chose GCP to save cost. We were not familiar with running Hadoop jobs on GCP so we used EMR for Hadoop Mapreduce.
- What are the most effective ways to speed up E & T?
  - ❏ Test code locally with small datasets to ensure correctness, and used parallel programming as much as possible.
- Did you use EMR? Streaming or non-streaming? Which approach would be faster and why?
  - ❏ Yes, we used EMR for Hadoop MapReduce jobs, which is non-streaming. Non-streaming could be faster for transforming data steps like aggregates hashtags with same keywords, since the sub-tasks are executed on different nodes in parallel.
- How did you load data into each database? Are there other ways to do this, and what are the advantages of each method? Did you try some ways to facilitate loading?
  - ❏ For MySQL, we loaded data into each database using a sql script. There are many other ways to loading data including creating database backups, creating an AMI or backup volumes. We tried all of them but encountered some problems with "http

socket read timeout" exception. We would look for solutions to fix this since manually loading data into each instance was really frustrating.

- ❏ For HBase, we loaded data into one instance and created a snapshot of database and stored on S3. After that, we restored database from the snapshot each time which was pretty fast.

Task 3: Databases

Question 7: Schema

For Query 2, we want to make changes to the raw data and then store it in databases in a good format, so that we can retrieve it fast. Various schemas can result in very large difference in performance! It is easy to come up with a working and slow schema, but it would be an order of magnitude slower than the target RPS. Then you might want to look at different metrics to understand the potential bottlenecks and come up a better solution (before trying to tune parameters!). These iterations take a lot of time and a lot of experimentation so start early.

- What different schemas have you designed and explored for each query and database? Which one did you choose in the end? Why?
  - ❏ We tested two different versions of schemas for Query2. For the first one we aggregated hashtags based on unique (keyword, uid) pair, and when the requests come in, we searched for all the records with matched keywords, and aggregated hashtags from all these records to get the final result. The second schema used two tables. The first table is exactly same as we used in the previous schema, and the second table aggregated hashtags based on unique (keyword). When the requests come in, we them 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 used the second schema in the end which improved our query performance a lot.

- How did the different schemas impact performance? It is a good idea to benchmark them on the same hardware, maybe a single instance. We expect you to show the different experiments you ran to justify your decisions.
  - ❏ The second schema gave us much better performance during our tests with hbase. We used exactly same cluster (a small one) to run this experiment, and the first schema yield a throughput around 110, while the second one gave 440.

- Try to explain why one schema is faster than another.
  - ❏ This difference is mainly because the first schema needed to do a 'scan' operation, while the second one only needed to do two 'get' operations, which was much faster. And also with the second shema we are doing some extra work before loading data into the database (aggregate hashtags with the same keyword), which also saved time for organizing results when handling the requests.

Question 8: Understanding the bottlenecks

It is fine if your servers ended up very far from our target RPS the first time. You have chosen a high-performance web frontend, but what other components in the system might be saturated and are pulling the performance of the service down? These questions will help you identify the sources, so you know what to think of when revising the schema.

- What is the capacity of all hardware components on your server VM? For example, CPU, memory, disk I/O, network, etc.
  - ❏ CPU: 2
  - ❏ Memory: 8 GiB
  - ❏ Network: m4.large: moderate/m5.large: Up to 10 Gigabit
- How do you monitor their utilization when the server is being tested? What numbers do you see and which capacity is saturated?
  - ❏ Use cloudwatch to monitor CPU utilization and network performance and disk I/O.
- Did you encounter this problem frequently with this DB (MySQL and HBase), and why? How did you solve it?
  - ❏ Yes, we encountered performance issue on MySQL and HBase, because the response time in the database is long. We ended up with applying various optimization and tweaks described in question 9 to achieve the target RPS.
- Explain briefly the theory behind at least 3 performance optimization techniques for databases in general. Are they related to the problem you found above?
  - ❏ Firstly, schema design should minimize the time to query, e.g. parsing string can be time consuming, so it is better not to store too long a string in one column.
  - ❏ Secondly, in sharding database, each server handles different amount of requests. We can observe the traffic pattern and manually divide up region server to balance requests handling in each server.
  - ❏ Thirdly, database configuration can be tuned, e.g. decreasing blocksize, modifying cache-size, set-up in memory storage or changing engine mode between read/write.

- 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)?
  - ❏ All of the optimization can be simulated in the other, since we have similar schemas for both database, except that we didn't use sharding in MySQL so we don't need to implement server division in MySQL. However, the configuration for MySQL and HBase is a little different.
  - ❏ MySQL: MySQL tables have engine types, it can be changed to read or write mode. Also, the connection pool of threads can be modified via jdbc configuration.
  - ❏ HBase: HBase configuration can be tuned in terms of block-size, cache-size and in-memory storage, etc. Get is much faster than Scan in HBase, to utilize this feature, we have two tables so that we used Get to query two times.

Question 9: Optimizations and Tweaks

Frameworks and databases do not come in the best shape out-of-the-box. Each of them has tens of parameters to tune. Once you are satisfied with the 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. No need to feel obliged to test all of them now. In future phases, you will continue to try to improve the performance of these queries.

Hint: It is advised to first focus on improving your schema and get an okay performance with the maximum number and type of allowed instances. If you are 10 times (an order of magnitude) away from the target, then it is very unlikely to make up the difference with these optimizations.

- List as many possible items to tweak as you can find that might impact performance, in terms of: web framework, your program, DBs, DB connectors, OS …
  - ❏ Web framework: try different frameworks
  - ❏ Program: apply cache to memorize previous request results.
  - ❏ HBase: query on two tables(<word, uid, hashtag> and <word, hashtag>), set in_memory as true; divide up regions manually; block size decrease;
  - ❏ MySQL: query on two tables(<word, uid, hashtag> and <word, hashtag>, use horizontal partition
  - ❏ DB connectors: increase max thread pool size
  - ❏ OS: increase the size of cache memory

- Apply one optimization at a time and show a chart with their respective RPS.
  - ❏ We did not try all of them, and we listed some of the optimizations we applied during our tests below:

Web Frameworks	For Q1: undertow gave > 27000 throughput, and vertx gave ~24000 throughput
Frontend cache	For Q2, cache improves overall throughput from ~3000 to > 6000, after several submissions
DB schema	For Q2, before any optimization, query from one table gave ~100 throughput, and query from two tables gave ~440 throughput
Connection thread pool size	For Q2 HBase, increase thread pool size helped when we were testing using a small cluster (1 master + 2 cores), but did not help with a large cluster (1 master + 4 cores)
OS memory	Did not help at all

- For every tweak, try to explain how it contributes to performance.
  - ❏ For web framework part, the connector XNIO of undertow intialized the number of logical threads and the worker thread equals to 8 times CPU cores, which is much

lower than Tomcats' and Jetty's default(100 and 1000 threads respectively). By increasing the threads, it could improve the throughput.

- ❏ For program part, with storing the previous retrieval results, we could save the time for retrieving the same query and improve the latency.
- ❏ A general strategy we used in both databases is to load two tables (<word, uid, hashtag> and <word, hashtag>) for each query to save time at calculating scores and improve latency.
- ❏ Set in memory as true in Hbase so that data is to be kept in cache for longest period possible to improve performance.
- ❏ With horizontal partition, we split the large table into smaller distinct parts(in our case, four subtables). It offers an advantage by reducing index size and provide a faster read throughput.

Task 4: Site reliability

Question 10: Monitoring

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

- What CloudWatch metrics are useful for monitoring the health of your system? What metrics help you understand performance?
  - ❏ CPU utilization, Network in/out.
- What statistics can you collect when logged into the VM? What tools did you use? What did they tell you?
  - ❏ Statistics: number of CPUs, memory, network performance, instance storage, CPU utilization, network utilization, disk performance, disk Reads/Writes. We used EC2 dashboard and cloudwatch dashboard.
- How do you monitor the status of each database? What interesting facts did you observe?
  - ❏ HBase UI: the status of region servers and the tables it hosts. Each region handles different loads of requests, and the difference between each region can be large.
  - ❏ MySQL command line tools such as Mytop, Mtop, Innotop and mysqladmin. Statistics: server configuration, current status, threads, process, queries, slow queries, uptime, load, etc.

Question 11: Profiling

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

- Given a typical request-response for Q2, describe the latency of each of these components:
  - a Load Generator to Load Balancer (if any, else merge with b.)
    - ❏ Load generator sends http requests to load balancer through network.
  - b Load Balancer to Web Service

        - ❏ Load balancer needs to distribute http requests among different frontend servers through network.
    c Parsing request
        - ❏ Frontend needs to parse request to get parameters and generate queries
    d Web Service to DB
        - ❏ Frontend needs to send queries through connection to backend databases.
    e At DB (execution)
        - ❏ Databases need to search tables to get results of queries.
    f DB to Web Service
        - ❏ Backend databases send query results to frontend servers.
    g Parsing DB response
        - ❏ Frontend servers parse each query result and aggregates all of them to get the final result of each request.
    h Web Service to LB
        - ❏ Frontend server send response to load balancer.
    i LB to LG
        - ❏ Load balancer send responses to load generator.
- For each step, think about how you can measure it, and write down the tools / techniques that you used.
    - ❏ The overall latency of load balancer could be retrieved from CloudWatch. The latencies of each step executed by frontend servers could be measured using tools like JProfiler. The latencies within DB could probably be measured using monitoring tools.
    - ❏ We did not do a very detailed profiling using fancy tools, but simply printed out the time of each step in frontend servers.

Task 5: General questions

Question 12: 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?
  - ❏ Since we are using replica with MySQL, our design may not meet the requirement of fast writing.

Question 13: 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 tested loads by submitting.
- Describe an alternative design to your system that you wish you had time to try.
  - ❏ Other designs we could have tried: MySQL sharding as described in Question 3.2; manually writing ELB for MySQL and connection pool for Hbase, since we want to configure ; change frontend framework to vertx, because bus event may better handle distributed system.
- Which was/were the toughest roadblocks that your team faced in Phase 1?
  - ❏ One of the toughest roadblocks is to reach target throughput. For both databases, we made several optimizations including trying different schemas, writing cache (though not well-designed), tuning configurations of instances.
- Did you do something unique (any cool optimization/trick/hack) that you would like to share?
  - ❏ No, we did not do have anything unique, and we appreciate any useful suggestions from TAs!!!

Question 14: Non-technical

- How did you divide components in the system? How do the components interact? How did you partition the work and what is each team member responsible for?
  - ❏ We divide the system into: Front-end, Database Design, initial Data filtering and Data Transformation with Hadoop MapReduce. Transformed data are then loaded to HBase and MySQL. Frontend servers receive HTTP requests, queries data from database, process the data from databases, and send response to clients. Long Wang is responsible for data filtering, processing and MySQL database; Xianting Wang is responsible for data processing and HBase database; Ke Lu is responsible for frontend.

- How did you test each component (unit testing and integration testing)?
  - ❏ We did unit testing on small functions and frontend servers. We tested ETL codes locally using small datasets as much as possible before moving to Cloud VMs.

- Did you like the Git workflow, code review, and internal testing requirements? Were they helpful to your development?
  - ❏ Yes, they are helpful.
- On TPZ, a new tab will show your Github contribution stats, and you need to include a screenshot of it in the Phase 1 report to reflect the contribution of each team member.