F16 15619 Project Phase 2 Report

Team Name: Craig300

Members (First Name, Last Name, Andrew ID):
Yifei Liu (yifeil3)
Yu Zhou (yuz5)
Xiaochen Wang (xiaoche1)

Performance Data and Configurations

<table>
<tr><th colspan="2">Best Configuration and Results</th></tr>
<tr><td>Number and type of instances</td><td>1 ELB
12 m4.large (3 for frontend, 4 for HBase and 5 for MySQL)</td></tr>
<tr><td>Cost per hour of entire system</td><td>$1.9</td></tr>
<tr><td>Queries Per Second (QPS) of your best submission</td><td>
<table>
<tr><td></td><td>Q1</td><td>Q2H</td><td>Q2M</td><td>Q3H</td><td>Q3M</td></tr>
<tr><td>score</td><td>5.69</td><td>9.01</td><td>2.06</td><td>N/A</td><td>0.00</td></tr>
<tr><td>submission id (for the above score)</td><td>41840</td><td>39960</td><td>41757</td><td>N/A</td><td>30629</td></tr>
<tr><td>throughput</td><td>5686.50</td><td>1590.30</td><td>246.50</td><td>N/A</td><td>13858.70</td></tr>
<tr><td>latency</td><td>16.00</td><td>30.00</td><td>199.00</td><td>N/A</td><td>3.00</td></tr>
<tr><td>correctness</td><td>100.00</td><td>68.00</td><td>100.00</td><td>N/A</td><td>0.00</td></tr>
<tr><td>error</td><td>0.00</td><td>0.01</td><td>0.00</td><td>N/A</td><td>0.00</td></tr>
</table>
</td></tr>
<tr><td>What is your team’s rank on the scoreboard:</td><td>49</td></tr>
</table>

Rubric:

Each unanswered question = -5%
Each unsatisfactory answer = -2%

[Please provide an insightful, data-driven, colorful, chart/table-filled, and interesting final report. This is worth a quarter of the grade for Phase 2. Use the report as a record of your progress, and then condense it before sharing it with us. Questions ending with "Why?" need evidence (not just logic)]

Task 1: Front end
Questions

1. Describe the design process and architecture of your front end system. Draw the architecture and provide experimental evidence as well as arguments to support your design decisions.

   Our front end was composed of 1 ELB and 3 m4.large VMs on which Vertx was deployed. We designed the project in this way to accomadate the needs of the live test because live test would test huge amounts of traffic as well as a mix test across three queries. ELB would distribute HTTP requests to the 3 servers running at its behind to handle the heavy traffic and improve performance. In each of the server, there was a set of logic to judge the request was for which query (Q1, Q2 and Q3) and query related backend database to retrieve the data.

   a. Which front end framework did you use? Explain why you used this solution. Did you change your framework after Phase 1? Why or why not? [Provide a small table of special properties that this framework/platform provides]

      Apart from ELB, we used Vertx as our real front end framework. There are three reasons behind it.

      1) Vertx is a high-performance web server which can help us reach designated QPS relatively easily.
      2) It is easy to program with Vertx because we can implement our front end by simply implementing an interface. This is much more convenient as compared to other frameworks.
      3) Vertx is a light-weight framework. It's easy to deploy and optimize it. Vertx also gets along well with Maven which we adopted to build our project.

      In Phase 2, we used Vertx which was the same as Phase 1. The reason is that the throughput of Vertx is satistiable and has been fully tested in Phase 1. Its simpliness and good performance make us continue using it.

      In addition, we deployed an ELB to handle the much heavier traffic which we didn't encounter for Phase 1.

   b. Explain your choice of instance type and numbers for your front end system.

      The front end system contains three m4.large machines. We used the same image created in Phase 1. In order to meet the throughput requirement and cost constraints, we also used an ELB to distribute HTTP requests to these machines.

      The reason why we chose m4.large instance was that it provides the best

performance under the restriction of limited budget. m4 instances have two cores which better support our multi-thread programming and they have a slightly larger memory than m3 series.

In each of our Vertx server, there are 4 threads to handle the incoming requests. We used 3 instances in light of the fact that we totally had 5 queries to test.

2. Explain any optimizations or special configurations used in your front end system. [provide experimental results or charts to support your decision as to which optimization provided improvements in performance]
   We basically used multithread in each of our Vertx server to improve our performance of front end system.

3. Did you use an ELB for the front-end? Why, or why not? Describe your experience with ELB in a few sentences. Discuss load-balancing in general, why it matters in the cloud and how it's applicable to your front end.

   We used ELB for the front-end. There were three front-end machines, so we created an ELB to distribute requests to these three machines to improve throughput.
   Load-balancing is crucial for cloud related applications since in the world, there would be enormous requests that exceed the capability of single machine. Although we can allocate these requests to different backend machines manually, it's difficult to achieve it equally for each node. Therefore load-balancing mechanism such as ELB will help us maximizing the power of front-end machines by evenly distribute request to each of the machines.
   In our case, since we have three front-end machines and each of them would use 4 threads to handle incoming requests. With the help of ELB, the total requests that can be handled is three times the amount of a single machine.

4. Did you explore any alternatives to ELB? List a few of these alternatives. What did you finally decide to use? (if possible) Provide some graphs comparing performance between different types of systems your team evaluated.

   Besides ELB, we also implemented hashing in the front end for MySQL part. For each request from the client, we hash the userId and forward it to one of our 5 back ends and each backend maintained entries with same hash code. In this case, the load was distributed into 5 and because the database is divided, query performance was improved as well.

5. Did you build your own custom AMI for your front-end instance? If yes, what was different in your AMI? E.g. did you pre-install some packages, did you have any services running at boot-time, did it automatically connect the web server to your backend, did it automatically trigger warm-up of your load balancer or EBS.

Yes. We created our own AMI using Ubuntu 14.04 image. We configured Java, Maven, Git and MySQL-server. We didn't start any service automatically during boot-time.

6. Did you use any form of monitoring of your front-end? Why or why not? If you did, show us a capture of your monitoring during the Live Test. Else, try to provide CloudWatch logs of your Live Test in terms of memory, CPU, disk and network utilization. Demarcate each query clearly in the submitted image capture.

   No, we didn't use any form of monitoring of our front-end. The reason for this was that monitoring tools would also consume system resources and negatively impacted the performance. We tried to eliminate all overheads in order to achieve as high performance as we could.

7. What was the cost to develop the front end system? You can use tagging to figure out the cost to develop and test your front end system.

   Since we used the same AMI we built in phase 1, the cost of development in phase 2 is minimal, less than 10 dollars.

8. What are the best reference URLs (or books) that you found for your front-end? Provide at least 3.

   1. https://aws.amazon.com/elasticloadbalancing/
   2. http://docs.aws.amazon.com/streams/latest/dev/kinesis-record-processor-scaling.html
   3. https://aws.amazon.com/articles/Amazon-RDS/0040302286264415

9. Please provide a comparison between at least two front end frameworks. Give explanations on at least the following dimensions: (If you have already done this well in Phase 1, just copy the result here)
   a. CPU/Memory utilization
   b. Programming difficulty
   c. What metrics did you use to choose one over another?
   d. What is your RPS for Q1 on all the frameworks you tested (this should be greater than 1 framework)

   The front end frameworks which we are going to compare here are Vertx and Tomcat and we will compare them in terms of programming difficulty and the metrics we used to choose one over another.

   Tomcat is a Java Servlet container upon which various web applications can be deployed. In order to use Tomcat, we need to build a web application (literally a WAR file) and deploy it. During the course of the development with Tomcat, we need to do configuration, coding, packaging and a complicated deployment.

On the other hand, Vertx doesn't have these drawbacks. We simply need to implement an interface. In addition, Vertx can be launched by running a Maven command. This will save us a lot of time because we used Maven to build our project.

In our case, Vertx is more favorable than Tomcat for the following reasons:

1) Vertex has lower programming and deployment difficulty than Tomcat.
2) Vertex is better at handling huge network traffic.
3) Vertex is lighter than Tomcat so that we could save a lot of time to focus on main tasks.

[Please submit the code for the frontend in your code ZIP file]

Task 2: Back end (database)
Questions

1. Describe your schema.
    a. Explain your schema design decisions.
    b. Would your design be different if you were not using this database?
    c. How many iterations did your schema design require?
    d. Also mention any other design ideas you had, and why you chose this one?
    e. Your answers should be backed by evidence (actual test results and bar charts).

a. As stated in report 1, the data structure we designed for phase 1 ETL is listed below:
userId, tweet1, timestamp1, impactScore1, tweet2, timestamp2, impactScore2 … , totalWordCount

The MySQL schema is defined as follow:

userId	*VARCHAR(64) NOT NULL PRIMARY KEY*
content	*VARCHAR(1024) BLOB,*
wordCount	*VARCHAR(1024)*

The HBase table is defined as below:
userId(HBASE_ROW_KEY), content, wordCount

We found this schema is not suitable for phase 2 query since the range query in Q2 is more time consuming than query in Q1. The design of content column in both MySQL and HBase requires a lot of parsing effort in front-end which will largely impact the performance of front-end.
Therefore, we designed new schema for phase 2 as below:
date+uid, tid, wordCount

The reason is that by combining the date and user id, we can use setStartRow and setStopRow for Scan operation
In order to improve the query performance, we use the last 5 digit of date plus the user id as rowKey for HBase. The reason is that we found all the earliest tweet was generated in the year 2010, which means there would be no duplication if only using the last digit of year, by doing so we can shorten the rowKey length to speed up the scan operation.

b. If we were able to use different database, the design of phase2 might be different. For instance, if we were allowed to use MongoDB, we can store all the tweets of user as an json object. The range query can be finished using MongoDB API, however in HBase, the wordCount is store as binary file which doesn't support query.
c. It took us two iterations to design our schema.
In the first iteration, we used below schema:
tweetId, userId, date, wordCount

We add filters during query, however we found the performance was not good since we add two filters.

Then we changed the schema to below:

date + userId + tweetId, wordCount

In this way, we could use one filter, but the rowKey is much longer this time and problematic.

Finally we decided to use schema:

date + uid, tid, wordCount

2. What was the most expensive operation / biggest problem with your DB that you had to resolve for Q2? Why does this problem exist in this DB? How did you resolve it? Plot a chart showing the improvements with time.

   The most expensive operation with our DB was the development of HBase System. After ETL, about 10 G dataset was created. To import the datase into the database, more cost is needed to increase the storage volum of the clusetr. After import the dataset into Hbase, we spent a mount of time to figure out the right operation to backup and restore the database system, which was time consuming and cost consuming. We also needed to restart our database each time we change the setting of database. In this case, the cluster needed to be rebuilt and the dataset needed to be restored from the backup and the test needed to run again. As we also tried to optimized the schema of the database, each time a new schema was redesigned, the whole operation was carried out another time.

3. Explain (briefly) **the theory** behind (at least) 3 performance optimization techniques for databases in general. How are each of these optimizations implemented in MySQL? How are each of these optimizations implemented in HBase? Which optimizations only exist for one type of DB and why? How can you simulate that optimization in the other (or if you cannot, why not)? [Always use your own words (paraphrase when necessary)].

   For HBase, we tried to changed default size of blockcache. Blockcache set to instore data blocks after after there are read. If the blcokcache size increased, more data block could be placed inside the memory, which could same plenty of time when the same block was recalled. In default, the size of blockcache was only 400Mb of the instance memory. As the instance memory is 8G, more of it could be spared for a larger blockcache. Also, to optimize the utilization of blockcache, the block size of each data entry could be minimized. As each entry only store the word count and userId of that tweet, the size is unliked to exceed 16 kb. In this case, default 64 kb block size seems prety large. After minized the block size, more block could be placed into blockcache, whick could help to

increase the hit rateof query.

4. HBase relies on HDFS, which is a fault-tolerant and distributed file system. How useful is it to have a distributed file system for this type of application/backend? Does it have any significant overhead?

   The advantage of distributed system as backend is to reduce the likelihood of congestion. One node can be overwhelmed if there's too many queries, using distributed system can reduce the workload of each node.
   Another advantage of distrbuted backend is that one datanode failure would not fail the entire backend system. If all the data is stored in one node, the node failure will make the whole system unusable.
   The overhead is that the front-end system has to coordinate between different backend nodes. There has to be a mechanism to decide which node the request should be sent to.

5. Plot a graph showing results with/without each individual optimization that you used. Extremely impressive will be a timeline of rps v/s submission id (mentioning which optimization was in use at that time).

   We didn't record related data in this regard, hense we are not able to provide a graph.

6. Would your design work if your web service also implemented insert/update (PUT) requests? Why or why not?

   Our design might work if the service also has to support insert/update(PUT) operation. The uncertainty is the performance since we might have to loop through wordCount if the insert/update needs to use this field.

7. Which API/driver did you use to connect to the backend? Why? What were the other alternatives that you tried? What changed from Phase 1?
   MySQL:
   We used JDBC to connect to our MySQL database in the backend. We didn't try any other alternatives because we used Java as our programming language and JDBC is of the highest efficiency and provides the least programming complexity in this situation. The change we made from Phase 1 was that we explicitly stated the charset we would use (UTF-8) as the parameters of the connection URL. We did that to ensure our retrieved data was encoded correctly.

   HBase:
   we used 'ConnectionFactory.createConnection' API to communicate with backend. We didn't try any other way to connect to DB. Nothing special changed in HBase part except we explicitly using UTF-8 encoding after retrieving data from HBase.

8. You should profile your backend to understand the source of a bottleneck. Describe how you profiled your backend databases?

   We used logging to detect the potential performance hit. We found that the low performance was caused by complicated parsing/sorting and formatting logic in frontend. Since we didn't use JSON to store the data, we had to do replacement special characters such as '\n','\t','\r',',' to avoid strange things during HBase importing. Thus we have to replace them back using replaceAll in the front end which largely affect the overall performance.

9. How can you reduce the time required to perform scan-reads in MySQL and HBase?
   MySQL:
   We basically took three actions to reduce time required to perform reads.
   First, we evaluated how different storage engines, i.e. InnoDB and MyISAM can impact the reads performance. After several testings, we found that MyISAM was more favorable in the case of reads. So we chose MyISAM as our storage engine.
   Second, we queried our data by "userId". "userId" is also our database's primary key which has an index built automatically. This index could help us improve performance.
   Finally, we used Sharding paradigm to distribute data into different databases running on different servers. We totally had 20G, 24 million records of data after ETL and they are distributed to 5 separate databases with each having 4G, 5 million records of data. This Sharding machanism significantly reduced the reads time.

   HBase:
   In order to reduce the time required to perform scan-reads in HBase, we designed several versions of rowKey. In phase1, we use tweet id as rowKey which is not sufficient enough for phase 2. Thus we trie to combine different fields to create new rowKey to improve performance by using setStartRow and setStopRow for Scan operation. We thought by combining two monotonously increasing fields, the rowKey can be used effectively by adding setStartRow and setStopRow.

10. Did you use separate tables for Q2 and Q3 on HBase and MySQL? How can you consolidate your tables to reduce memory usage?

    For HBase, we designed separate tables for Q2 and Q3.
    Q2 schema is : *userId(HBASE_ROW_KEY), content, wordCount*
    Q3 schema is: *date + uid (HBASE_ROW_KEY), tid, wordCount*
    However, we failed to import data to different HBase tables using ImportTsv. The latter imported one would eliminate the previously restored table.

11. We expect you to know the end-to-end latency of your system. Given a typical request-response for each query (Q1, Q2 & Q3), describe the latency of each of

these components (ignore d-g for Q1):

a. Load Generator to Load Balancer (if any, else merge with b.)
b. Load Balancer to Web Service
c. Parsing the request
d. Web Service to DB
e. At DB (execution)
f. DB to Web Service
g. Parsing DB response
h. Web Service to LB
i. LB to LG

[How did you measure this? A 9x2 table is one possible representation.]

We measured e and g for HBase. For Q3 we found the query was extremely slow (several seconds), because the setStartRow and setStopRow would return lots of results even if we add extra column filter logic.
During parsing, we would explicitly extract the wordCount object and add up the numbers. Then build the response by replacing special characters. This process was time consuming.

12. What was the cost to develop your back end system?

The overall cost of developing backend system was about 30-35 dollars.

13. What were the best resources or references (online or otherwise) that you found.
Answer for both HBase and MySQL.
MySQL:
https://dev.mysql.com/doc/refman/5.5/en/optimizing-myisam-bulk-data-loading.html
http://derwiki.tumblr.com/post/24490758395/loading-half-a-billion-rows-into-mysql
http://dev.mysql.com/doc/refman/5.7/en/optimization.html

HBase:
http://hbase.apache.org/0.94/book/client.filter.html
http://hbase.apache.org/0.94/book/important_configurations.html#recommended_configurations
http://hbase-perf-optimization.blogspot.com/2013/03/hbase-configuration-optimization.html

[Please submit the code for the backend in your code ZIP file]

Task 3: ETL

1. For each query, write about:
    a. The programming model used for the ETL job and justification
    b. The number and type of instances used and justification
    c. The spot cost for all instances used
    d. The execution time for the entire ETL process
    e. The overall cost of the ETL process
    f. The number of incomplete ETL runs before your final run
    g. Discuss difficulties encountered
    h. The size of the resulting database and reasoning
    i. The size of the backup

    In order to maintain the consistency along the whole project, Java was used to help decipher X and response the result. After phase 1, about 20G data was extracted from the 1TB raw data. As the size of the dataset had been reduced significantly, there is no need to use any memory optimized or computational optimized instance. 4 m4.large instance were selected to process ETL job. To save budget, $0.03 was set as the spot prices for each machine, which is ⅓ smaller than the market machine ¾ smaller than the on-demand price. The whole job cost around 45 minutes to process and it about $0.5 for each run. We have ran 3 times to finalized our final output. The most difficult problem we have met is loading data into the datase, whcih will be discussed in next question . After the whole datase is improted into both databases, backups were created for future use. The size of each back up is around 10G.

2. What are the most effective ways to speed up ETL? How did you optimize writing to MySQL and HBase? Did you make any changes to your tables after writing the data? How long does each load take? How can you parallelize loading data into MySQL?

    During the map-reduce process, the most effective way to speed up ETL was to select more power instance. However, in this phase, there was no need to select some more powerful instances as the primary dataset created in last phase could used to develop the database in this phase. For some reasons the loading process was instable for both HBase and MySQL. When we were loading data to HBase, the map-reduce process sometimes stuck at some stage and then reset the whole process. This problem was resolved by increase the volum of the cluster. For MySQL databse, we initiallyy use innodb engine to import the datase. However, the process was really instable. Sometimes it cost 10 minutes to import one part of the dataset, but sometiems it cost around 2 hour to import the same part into the database. After we change our storage engine to MyISAM, the import process was stable and efficient. As we used streaming program in EMR, the data was not parallezied loaed into MySQL.

3. Did you use EMR? Streaming or non-streaming? Which approach would be faster and why?

Yes, we used EMR with streaming program. Technically, non-streaming program could be faster as it could save time to load data into databases. The dataset could be imported into databases during the reduce process.

4. Did you use an external tool to load the data? Which one? Why? If you were to do Q2 (Phase 1) ETL again, what would you do differently?
   No, we did not use external tool to load the data.

5. Which database was easier to load (MySQL or HBase)? Why?
   HBase should be easier to load. The basic idea is because HBase could help to automatically manage the dataset, divided into different regions which could help to imporve its query performance. however, in MySQL, we have to do sharding by our selves.

[Please submit the code for the ETL job in your code ZIP file]

Task 4: General Questions

1. Would your design work as well if the size of the data set would double? What if it was 10 times larger? Why or why not?

   In this case, the datase is around 10G, if it is doubleed, the design could matain the same. However, if the datase is 10 times larger, the design might be reproduced. It is believed that database systems as MySQL, HBase could handle large dataset that is larger than the given one. However, to speed up the performance of the system and reduce latency brought from massive data process, more efficient data structures and query algorithms need to be developed and more powerful hardwares need to help support the system

2. Did you attempt to generate load on your own? If yes, how? And why?
   Yes, we tried to created a simple load generator to help to warm up ELB, before each test. This was carried out by implemneting the basic URL package and InpuStreamRead package in Java. A thread pool with 20 threads were created to concurrently send requests to ELB.

3. Describe an alternative design to your system that you wish you had time to try.
   As describe in privious sections, the optimization of HBase and MySQL need to be explore more in future phase. The change of blockcache as well as region division in HBase could potentially help to improve the system performance.

4. Which was/were the toughest roadblock(s) faced in Phase 2? How was it different from the issues in Phase 1?
   The toughest roadblcoks we have met in Phase 2 was performance of backend dabases. As we need to to make up the work in phase 1 as well as complet this query, time was really limited so that we did not have enough time to try different dirrections. Lacking of cominucation with instructors, sometimes it was really hard for us to realize how wrong we have progressed, and thus we were really frustrated and could not find the right way to do the project.

5. Did you do something unique (any cool optimization/trick/hack) that you would like to share with the us?

   Loading data into MySQL was a pain. At the beginning, we waited for 5 hours but to see the loading having no tendency to end. So we manually terminated the loading. At that time, we tried to load the whole dataset of 20G, 24 million records of data. After doing some research, we found that the storage engine was playing some role. We changed our storage engine form InnoDB to MyISAM and it typically took 15 minutes for the 24 million records of data to be loaded. This actually saved us a lot of development and testing time.

[NOTE:]

- **IN YOUR SUBMITTED ZIP FILE, PLEASE DO NOT PUT MORE ZIP OR GZ FILES (NO NESTED ARCHIVES)**
- **USE A SIMPLE DIRECTORY**
    - **/etl**
    - **/frontend**
    - **/backend**