F15 15619 Project Phase 1 Report

Team Name: coding squirrels

Members (First Name, Last Name, Andrew ID):
Haibin Lin (haibinl), Li Chen (lchen1), Hechao Li (hechaol)
Performance Data and Configurations

Best Configuration and	Results
Number and type of instances	Q1: 1 m4.large Q2H: Failed to test Q2M: Failed to test
Cost per hour	Q1: 0.019/hour; Q2H and Q2M: Failed to test
Queries Per Second (QPS)	INSERT HERE: (Q1,Q2H,Q2M) score[132.31, 0 0] tput [33078.6, - -] latecy [2, - -] corr [100.00, - -] error [0.00, - -]
Rank on the scoreboard:	Q1: 13 Q2H: - Q2M: -

Rubric:

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

[Please provide an insightful, data-driven, colorful, chart/table-filled, <u>and interesting</u> final report. This is worth a quarter of the grade for Phase 1. 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. **Which front end framework did you use? Explain why you used this solution. [Provide a small table of special properties that this framework/platform provides]**

Table I Choices of Frameworks

Framework		Vert.x 3
Features	Lightweight	Vert.x core is around 650kB in size.
	Fast	It ranks high at several tests in techempower ranking website (in previous years).
	Scale	Its Non-blocking I/O design can handle concurrent I/O operations using a small number of kernel threads.
	MySQL/HBase support	It supports Java JDBC package and Hbase Java API
	Documentation	It is well documented, with example usages.

We also looked at

1) Netty, which is also pretty light weighted with high throughput. But it doesn't come with detailed examples and documentations, which is why we didn't go with it.
2) Play! framework, but the performance was not good (under 10,000 rps).

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

Table II Choices of Instances

Instance Profile		Reasons
Instance Type	m4.large	According to https://aws.amazon.com/ec2/instance-types/, the Instance Type Matrix shows that m4.large instances provide the Enhanced Networking feature comparing to m3.large. Comparing to other instances under M family, m4.large provide more vCPUs or more memories. For these reasons , we use m4.large for its better performance.
Instance Numbers	1	The CPU utilization percentage ranges from 30% to 70% (shown in Figure 1) when testing front end services in m4.large, thus we don't need extra instances to share the load.

3. **Explain any special configurations of your front end system.**
   1) Install Maven;

2) Instal Java 8 and setting JAVA_HOME environmental variable.

4. **Did you use an ELB for the front-end? Why, or why not? Condense your experience with ELB in the next few sentences. Talk about load-balancing in general and why it matters in the cloud.**

   We didn't use ELB for the front-end. As mentioned in Question 2, The CPU utilization percentage only ranges from 30% to 70% (shown in Figure 1), it is not necessary to distribute the load to other instance. Thus, a load balancer is not needed.

   Besides, we did some experiments with ELB attached by 2 m4.large comparing to without ELB using only 1 m4.large instance, the CPU utilization of each instance attaching to ELB is less than 15% (shown in Figure 2), which is much less than that of the instance without ELB.

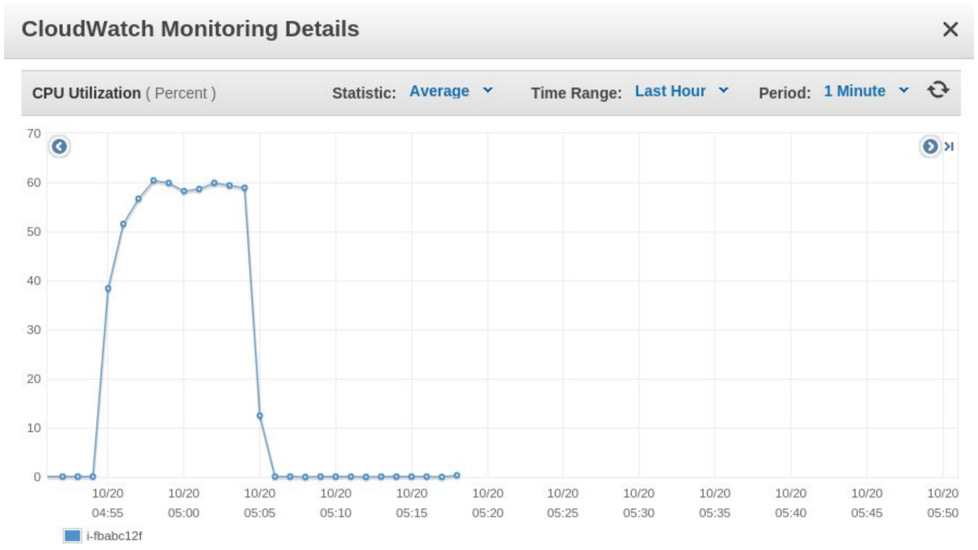


Figure 1 m4.large instance front-end test without ELB

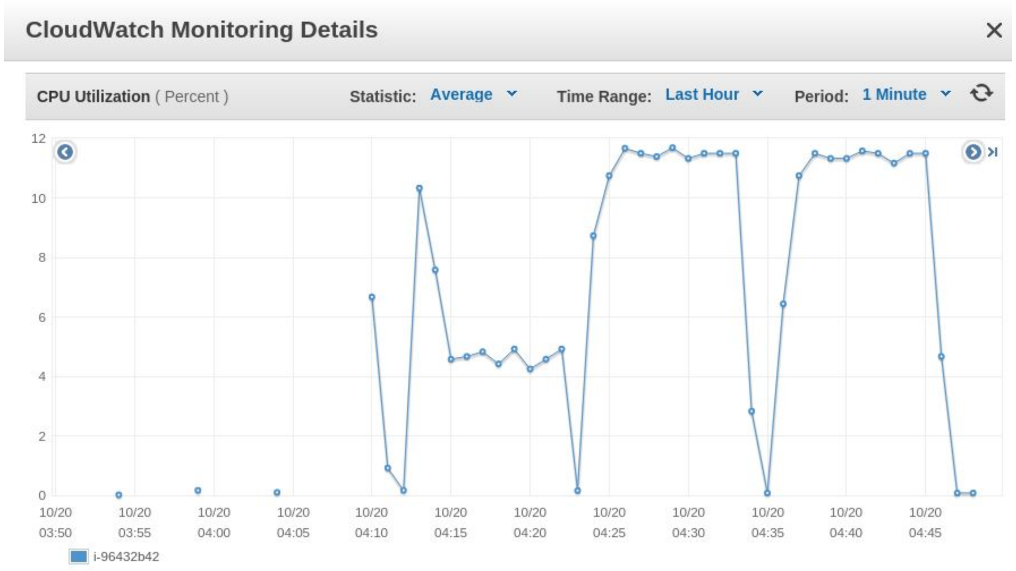


Figure 2 m4.large instance front-end test with ELB

ELB automatically distributes incoming application traffic across multiple instances that are attached to it. My experience is that in order to balance requests among instances as well as keep up the rps, we need to warm up ELB. This is because ELB also needs to scale up and down according to the traffic in order to handle traffic distribution. It is very important in the cloud because without it, traffic cannot be properly distributed to different instances in order to balance the traffic load as well as prevent from sending requests to failure instance.

5. **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.**
   No, we did not, since one instance is good enough to reach the target rps.
6. **Did you automate your front-end instance? If yes, how? If no, why not?**
   Yes, we set up an AMI based on the instance that we successfully configured. Whenever new instance is needed, instances are provisioned based on our AMI, in which every related configuration is done and web service can be automatically boot up by typing bash command in .bashrc profile.
7. **Did you use any form of monitoring on your front-end? Why or why not? If you did, show us the results.**
   Yes, we use the detailed cloud watch on monitoring the CPU utilization of our instances while testing the front-end services. Data can be seen in Figure 1 and 2.
8. **What was the cost to develop the front end system?**
   Java JDK 8 needs to be installed, Maven needs to be installed, Framework manual needs to be referenced for development. However, using frameworks really saves a lot of time to deploy front end.
9. **What are the best reference URLs (or books) that you found for your front-end? Provide at least 3.**
   1) Vert.x 3 Core Manual:
      http://vertx.io/docs/vertx-core/java/#_writing_http_servers_and_clients
   2) Vert.x 3 Example:
      https://github.com/vert-x3/vertx-examples
   3) Best Practices in Evaluating Elastic Load Balancing:
      http://aws.amazon.com/articles/1636185810492479

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

Task 2: Back end (database)
Questions

1. **Describe your schema. Explain your schema design decisions. Would your design be different if you were not using this database? How many iterations did your schema design require? Also mention any other design ideas you had, and why you chose this one? Answers backed by evidence (actual test results and bar charts) will be valued highly.**

Table III Schema Overview

MySQL	**Fields**	id	tid	uid	score	timestamp	epoch	text
	Fields Description	Auto Increment Primary Key.	Tweet ID	User Id	senti-ment score	time stamp value in format like "2014-05-31+01:29:04"	the number of seconds	text in format of json
HBase	**Rows**	uid+tid+timestamp (without "+") like "12408871992014-05-15+09:02:21466866162106511360"						
	Column Family	data						
	Columns	original text	tid	uid	score	timestamp	epoch	text

For hbase, we define key as uid_time_tid and just one family. The columns in the family are censored tweets, timestamp(string), sentiment score and tweet id. We also store the integer value of time in case we need to use it in the future for other queries, since integer is easier to compare. The timestamp here is in the same format of the incoming request.

The key is designed this way because Hbase use sharding. This way we can find all row with uid and time as prefix faster.

In MySQL, the columns are similar.

One thing to note is that we store text as JSON strings because we need to escape special characters during ETL, instead of storing plaintext directly. This also makes it easier to load the data.

The sentiment score and censored text are prepared during ETL.

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 text contains too many unpredicted characters in various coding scheme, such as Utf-8, Utf-16, Unicode, etc. And some special characters like '\t' and '\n' may truncate the text when loading data to the DB. The problem that '\t' will truncate the text exists in HBase because HBase only support '\t' as the delimiter. And the problem that unicode characters, which are in format like '\u0423', will become 'u0423' exists in MySql because MySql emerged before Unicode has been applied. So its support for Unicode is very poor.

For the problem in HBase, we wrap the text in a json string so that '\t' and '\n' can be escaped.

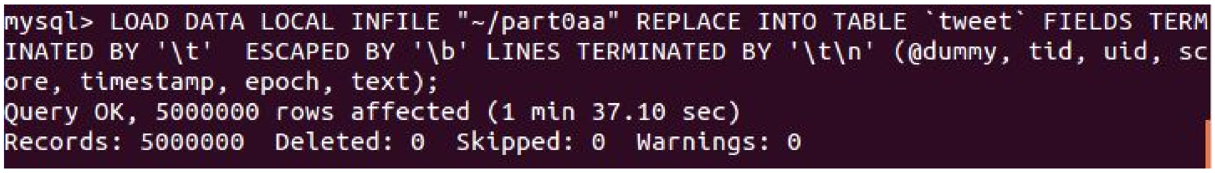
For the problem in MySQL, we load data into MySQL using '\t' as delimiter and '\t\n' as line terminator, also using '\b' as escape sign instead of default '\'.

3. **Explain (briefly) <u>the</u> <u>theory</u> behind (at least) 3 performance optimization techniques for databases. How are each of these implemented in MySQL? How are each of these implemented in HBase? Which optimizations only exist in one type of DB? How can you simulate that optimization in the other (or if you cannot, why not)? Use your own words (paraphrase).**

1) Create indexes: When creating an index, the database server will automatically measure and store the corresponding statistical information regarding the distribution of values in the indexed column. This statistical information is used by the optimizer to determine the optimal strategy for evaluating a query. It is implemented in MySQL but not implemented in HBase. For simulating in HBase, it is not necessary because HBase is searching values based on keys, which inherently indexed already.
2) Combine searching fields: For HBase, since it uses sharding, the design of the row key is very important. Since the records are stored in sorted order of row keys, we decided to use userId_timestamp_tid as the row key. This way, when querying all the tweets sent by a user in certain time, only a few consecutive rows are returned, which reduces the time for lookup. For MySQL, combining userId and Timestamp (which are two query values) is also a viable way to improve indexing efficiency.
3) We intentionally add an "id" field with auto_increment attribute when creating and loading data for MySQL, which increases loading data efficiency a lot (previously loading 1 GB data needs 20 mins, with `id` field setting to primary key instead of setting other fields as primary key, loading 1 GB only needs less than 2 mins). It is only used in MySQL. For simulating in HBase, It is not applicable since the row key shall be used for query, while in MySQL we can use other fields for query instead of `id` field.

4. **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). (lchen)**

For 3.(2), part0aa is Figure 3 is 1.3G.



```
mysql> LOAD DATA LOCAL INFILE "~/part0aa" REPLACE INTO TABLE `tweet` FIELDS TERM
INATED BY '\t'  ESCAPED BY '\b' LINES TERMINATED BY '\t\n' (@dummy, tid, uid, sc
ore, timestamp, epoch, text);
Query OK, 5000000 rows affected (1 min 37.10 sec)
Records: 5000000  Deleted: 0  Skipped: 0  Warnings: 0
```

5. **Would your design work if your web service also implemented insert/update (PUT) requests? Why or why not?**
   Yes. Because JDBC supports concurrency, and right now we are using only one instance for back end database, the insert/update operation will be thread-safe to ensure the data consistency .

6. **Which API/driver did you use to connect to the backend? Why? What were the other alternatives that you tried?**

JDBC, HBASE API. Also tried Happybase to connect with HBase. But that is a Python library, so we didn't use it for our web server.

7. **How did you profile the backend? If not, why not? Given a typical request-response for each query (q1-q2) what percentage of the overall latency is due to:**
   a. Load Generator to Load Balancer (if any, else merge with b.)
   b. Load Balancer to Web Service
   c. Parsing request
   d. Web Service to DB
   e. At DB (execution)
      (biggest)
   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 implemented a Profiler java class to capture the average delay in between events.

	q1	q2
a.b.c.	3 ms	4 ms
d.	0 ms	265 ms
e.	7 ms	400 ms
f.	0 ms	20 ms
g.h.i.	0 ms	0 ms

8. **Say you are at any big tech company (Google/Facebook/Twitter/Amazon etc.).**

List one concrete example of an application/query where they should be using NoSQL versus one where they should be using an RDBMS. Both examples should be based on the same company (you choose).

Twitter should be using NoSQL when a user open his/her home page and load latest tweets. Since this is a read-heavy request and does not need strong consistency but require low latency because in general a user do not care about whether the tweets he/she reads is the latest as long as there are something new to read. And users hope to read new posts as fast as possible. Besides, the tweets data is semi-structured and the schema may change with time and with users' demands.

Twitter should be using an RDBMS when a new user signs up. Because the schema of user profile is relatively fixed and structured. And user data needs strong consistency.

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

   There are so few documents of HBase that we need to spend much time searching for an answer. Besides, how to deal with characters in various charset is a huge problem which is time consuming. We spent around $10 to test the connectivity with HBase and MySQL and check if the data is correct.

10. **What were the best resources (online or otherwise) that you found. Answer for both HBase and MySQL.**

    HBase: official 0.94 doc, client connection part
    MySQL: jdbc official doc

[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**
    MAP REDUCE, MAP extract fields that need, REDUCE formatting
    b. **The number and type of instances used and justification**
    Use 1 m3.xlarge as master and 4 m3.xlarge as cores.
    c. **The spot cost for all instances used**
    0.0031 USD per t1.micro
    0.0166375 USD per m1.large
    0.0191117647 USD per m4.large
    0.03205 USD per m3.large
    0.0405554913 USD per c3.xlarge
    0.0410571429 USD per m3.xlarge
    0.043145 USD per c3.xlarge
    0.0560222222 USD per m4.2xlarge
    0.1013666667 USD per m3.2xlarge
    d. **The execution time for the entire ETL process**
    Around 14 hours per run. since we didn't make the encoding right, it was repeated several times and took a lot of time
    e. **The overall cost of the ETL process**
    $33.54. Most of them are spent one EMR's.
    f. **The number of incomplete ETL runs before your final run**
    Failed times : 25 - 26
    g. **Discuss difficulties encountered**
        1) When loading data to HBase, the delimiter cannot be user-defined.
        2) Loading terabytes of data takes a lot of time. We didn't do it well since our ETL process is not well tested, problems such as encoding got into our way and we had to repeat the process again and again.
        3) Creating Index in MySQL take a lot of time and some space. Before that, we should have done thorough checks on data for correctness!

    h. **The size of the resulting database and reasoning**
    About 50G, only part of the original one. It's much smaller than the original dataset since we have filtered out all data before 20th Apr, and we're only interested in certain fields of the data record.
    i. **The size of the backup**
    We made a backup for our data but it turned out that the text is not correct, so this backup was abandoned.

2. **What are the most effective ways to speed up ETL?**
    1) Use larger and more instance with higher performance, on-demand price.
    2) Before doing ETL on the whole data, try some small dataset and see if the logic of

ETLis right.

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

   Yes. Streaming is used. We don't have time to try non-streaming right now, so we don't know which one is faster.

4. **Did you use an external tool to load the data? Which one? Why?**

   For Hbase, we first tried happybase, a python library which helps us to insert data into Hbase. However, it doesn't work well when submitted as a mapreduce job and it hangs when the volume of data to insert is large.

   Hbase: We use importTsv to load data. We have tried to import data to Hbase during reducing, but that made the reduce process fairly slow. So we changed the strategy -- the reducer outputs .csv files and we load data to Hbase using importTsv.

5. **Which database was easier to load (MySQL or HBase)? Why?**

   MySQL is easier to load. Because the document of mysql is relatively more complete so that we can find official solution quickly when encountering problems. Besides, it is easier to spot warnings and check the mistaken places. In contrast, it is hard to find both official and unofficial solutions of HBase problems. And we are always distracted by too many logs that HBase prints.

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

General Questions

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

We're using only one instance for mysql currently. When the quantity of data increases several times larger, just one instance cannot handle such amount any more. The request will be blocked because of limited memory and computation power on the instance. In that case, we should try to replicate mysql into several instances so that each of them help handle some amount of request.

2. **Did you attempt to generate load on your own? If yes, how? And why?**

   We have not done it yet, but we plan to. We would use some popular load-generating tools such as Apache Bench, Httperf, Apache Jmeter, Tsung, etc. We will do some test on these tools and find a best one.

3. **Describe an alternative design to your system that you wish you had time to try.**

We can change our existing schema to make it more customized for query2 and improve the performance. For example, instead of storing data in separate fields, we can save the answer in just one field, containing the result for requested user id and timestamp. For example, we store the result of user with uid and timestamp t as uid_timstamp, {"result":{tid1:score1:text1\ntid2:score2:text2}}
Also, since we have a budget of $1.25 per hour, we can potential use more than just one instance running mysql database. Also, we can have a load balancer connected with 2 web front-end server to boost up performance.

4. **Which was/were the toughest roadblock(s) faced in Phase 1?**
   Data cleansing is the most difficult barrier for us during Phase 1. It is because:
   1) The escape sign '\' will be translated by MySQL when "ESCAPED BY" is not designated;
   2) The escape sign may exist at the end of the text to possibly ecape our quote symbol for our json-formatted text;
   3) the delimiter and new-line should be properly handled to be able not to collide with those within text field;
   4) unicode should be modified when loading into the database so that it can be recovered when query in database, This includes different symbols like different language symbols, facial expression symbols and so on.
5. **Did you do something unique (any cool optimization/trick/hack) that you would like to share with the class?**

   No