F15 15619 Project Phase 3 Report

Performance Data and Configurations

Live Test Configuration and	Results
Instance type	m3.large
Number of instances	8
Cost per hour	0.133(on-demand)*8+0.025(elb)+0.016*8=$1.22
Queries Per Second (QPS)	q1/q2/q3/q4/q5/q6/MIX[q1/q2/q3/q4/q5] INSERT HERE: score[6.25/4.8525/15/9.27/15/0/MIX[3.75/0.2835/0.27975/0.316125/3.75]] tput[37229.9/3301.8/9994.6/4943.6/22116.1/24950.3/MIX[5241.6/226.9/223.9/252.8/3000.4]] latcy [2/14/4/9/2/1/MIX[19/223/225/200/16]] corr [100.00/98.00/99.00/100.00/100.00/0/MIX[100.00/100.00/100.00/100.00/100.00]] error [0.00/0.01/0.00/0.00/0.00/0.13/MIX[0.00/0.00/0.00/0.00/0.00]]
Relative Rank [1 - 80] :	Phase 1 : 51 Phase 2 : MySQL 54 HBase 40 Phase 3 : 47
Phase Score [out of 100]	====== graded ====== Phase 1: 114.14 Phase 2 Live HBase: 19.18 Phase 2 Live MySQL: 20.32 Phase 3 Live: 58.75 ====== others ====== Phase 2 (Pre-Live): 218.33 Phase 3 (Pre-Live): 477.59

Team : Upinthecloud
Members :Silun Wang(silunw), Di Xiao(dxiao1), Yuwei Zhang (yuweiz1)

[Please provide an insightful, data-driven, colorful, chart/table-filled, humorous and

interesting **final report. This is worth 20% of the grade for Phase 3. Make sure you spend a proportional amount of time. For instance, if you spent 30 hours building your system, you should spend 10 hours on the report. The best way is to do both simultaneously, make the report 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]

   We use Vert.X. as our front end framework, as it provides several good properties listed below:
   1. use Java as programming language
   2. lightning fast and high throughput
   3. event driven and non-blocking

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

   We choose m3.large as our front end/back end instance. We put frontend and backend database on the same instance, and use an ELB to distribute load. As we are not allowed to use instances larger than 'large' type and only in 'm' family, we choose m3.large. m3.large has 2 CPUs and 7.5G memory, which can satisfy our demand of using memory to store data and handle request. Also, m3.large behaves better than m1.large.

3. Explain any special configurations of your front end system.

   No special configurations.

4. What did you change from Phase 1, 2 and why? If nothing, why not?

   We use one machine as the front end and 8 machines as backend in phase 2, but now we are using ELB to distribute requests to servers and each machine is a combination of frontend and backend.

   Using one front end and several backend will lead to high network and latency. Fetching data from the same machine will be faster. Also, in live test of phase 2, we face the problem that the only one frontend usually reaches 100% utilization, and when it failed due to huge load of requests and we didn't find out, we got zero point on the test.

   So we decided to use ELB to distribute request this time. During the live test, although we still have a huge amount of request, the average CPU utilization of each server is around 50% and each server can be used efficiently without overloading.

5. Did you use an ELB for the front-end? Why, or why not? Condense your experience with ELB in the next few sentences.

Yes.

An ELB will distribute requests evenly to servers and will make full use of each instance while not overload them. We used only one frontend instance last time and get 100% CPU utilization and lead to crash of the frontend server. Using an ELB this time leads to around 50% of CPU utilization of each server.

The most important thing to notice about using ELB is that ELB needs to be warmed up, so it can handle more requests and send more requests to servers, which will lead to high RPS. In our experience, RPS increased steadily when we run test on servers several times.

6. 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.

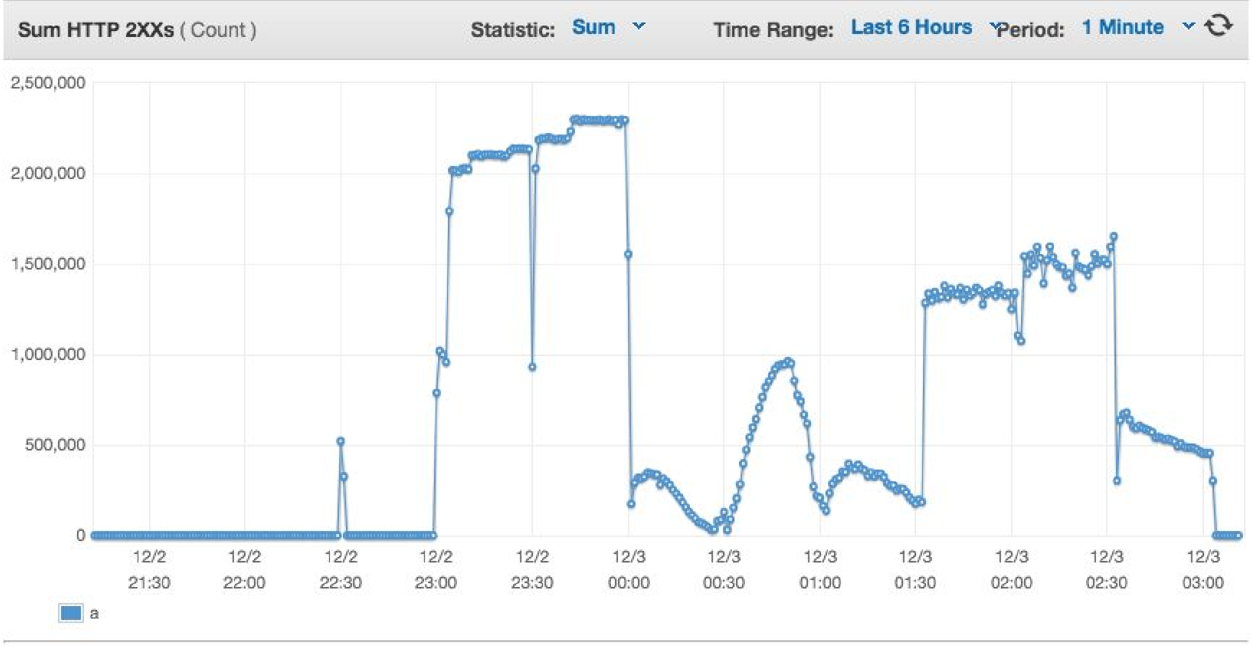
   We tried Nginx. But its performance was no better than using ELB, so we finally decided to use ELB.

7. Did you automate your front-end? If yes, how? If no, why not?

   We automate our front-end instances using maven.

8. Did you use any form of monitoring on your front-end? Why or why not? If you did, show us the results.

   We used AWS CloudWatch for ELB and individual instances. Monitor healthy status of each front end servers. We look at metrics like HTTP 2XX, 5XX and latency. On each front-end server, we monitor the connections and CPU, memory consumption. On each back-end server, we monitor the connections and execution time of each query.



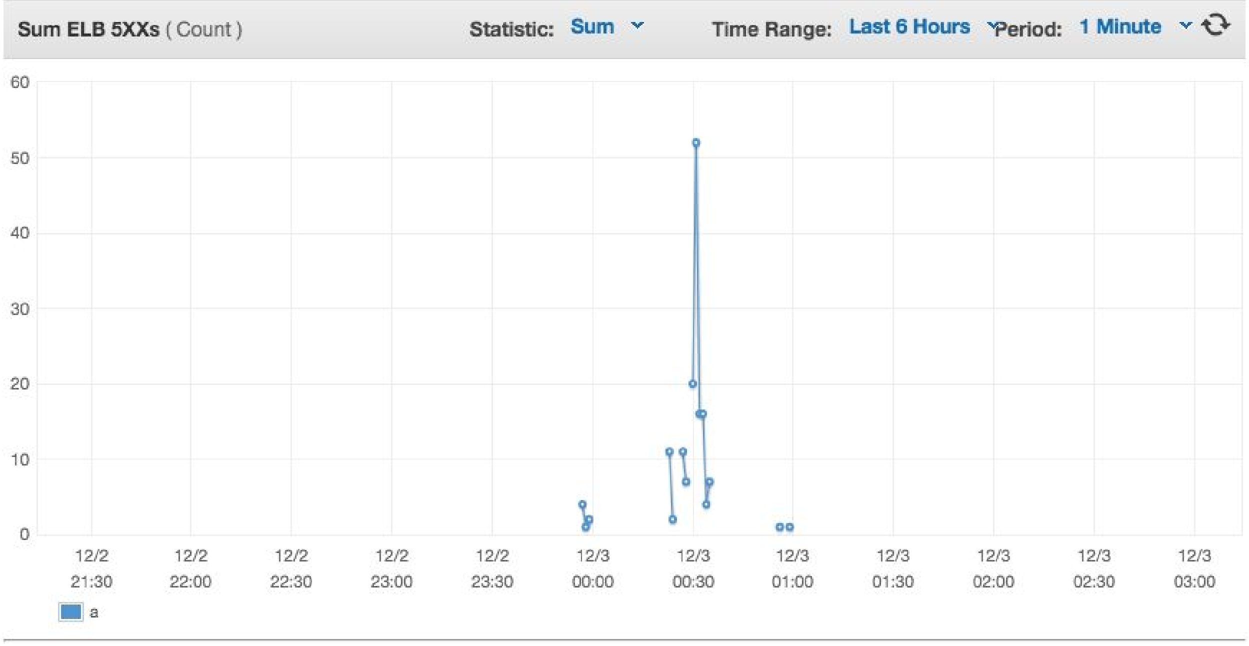
Sum HTTP 2XXs (Count)

Statistic: Sum

Time Range: Last 6 Hours

Period: 1 Minute

Warm-up | Q1 | Q2 | Q3 | Q4 | Q5 | Q6 | MIX

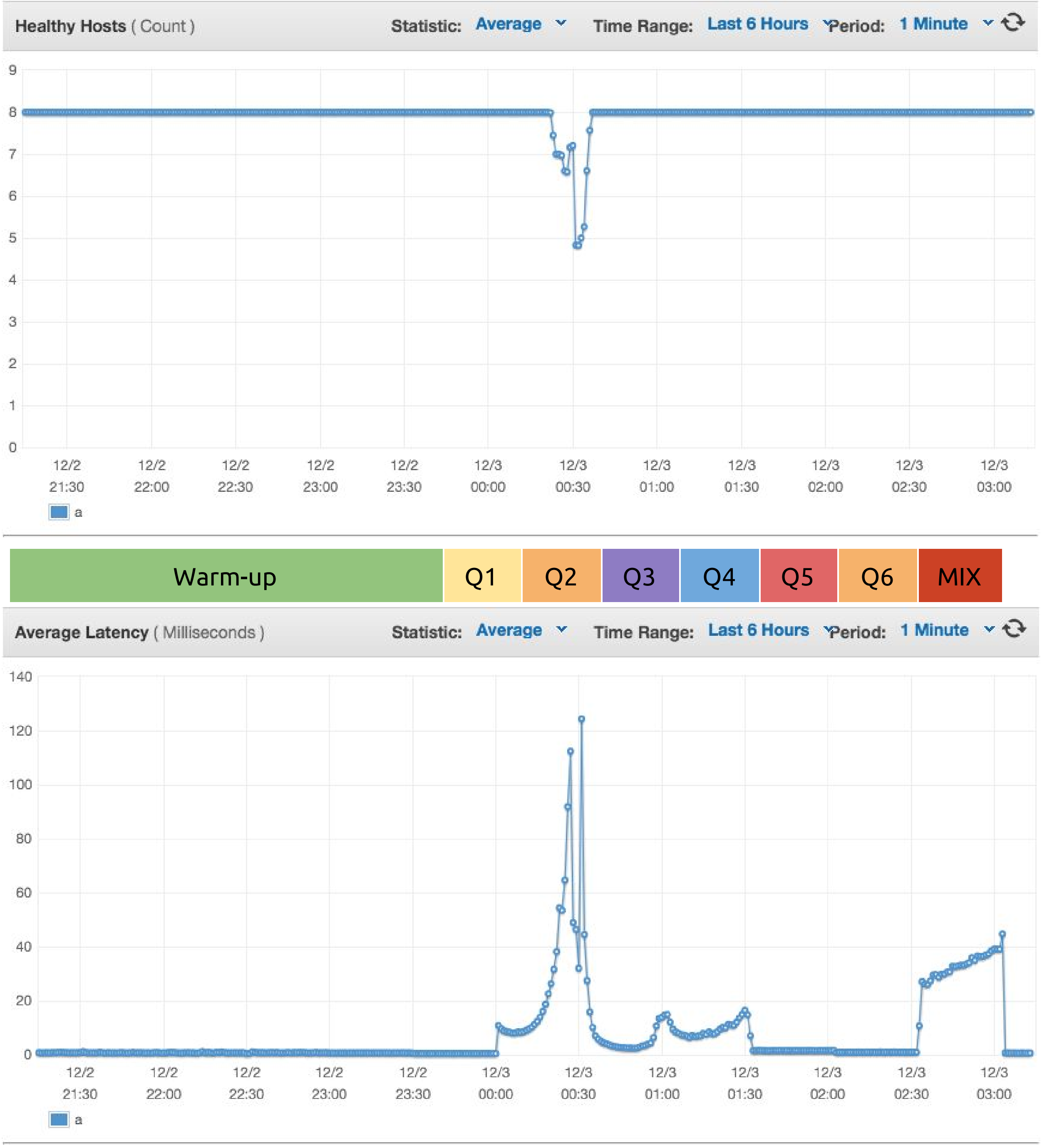


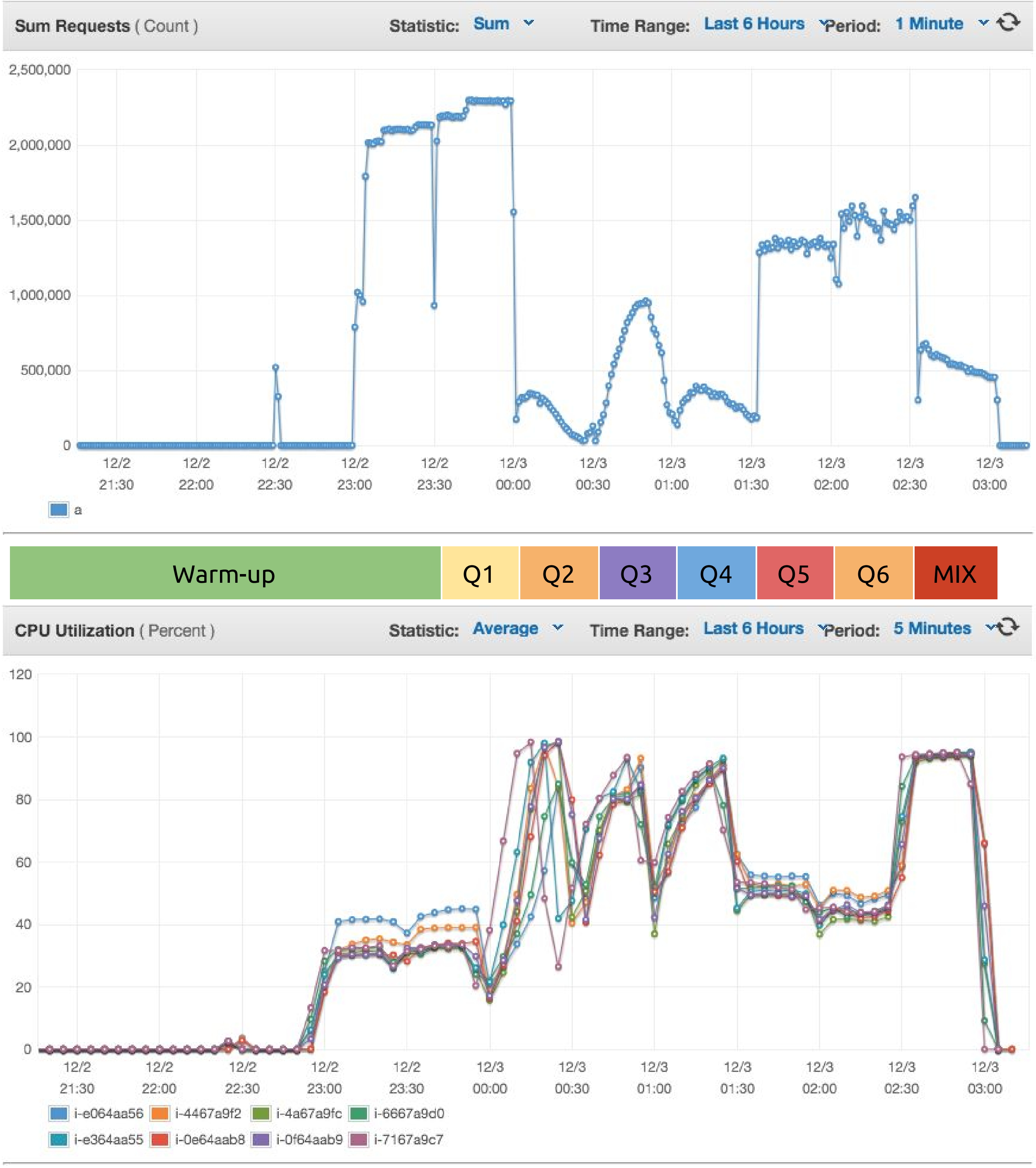
Sum ELB 5XXs (Count)

Statistic: Sum

Time Range: Last 6 Hours

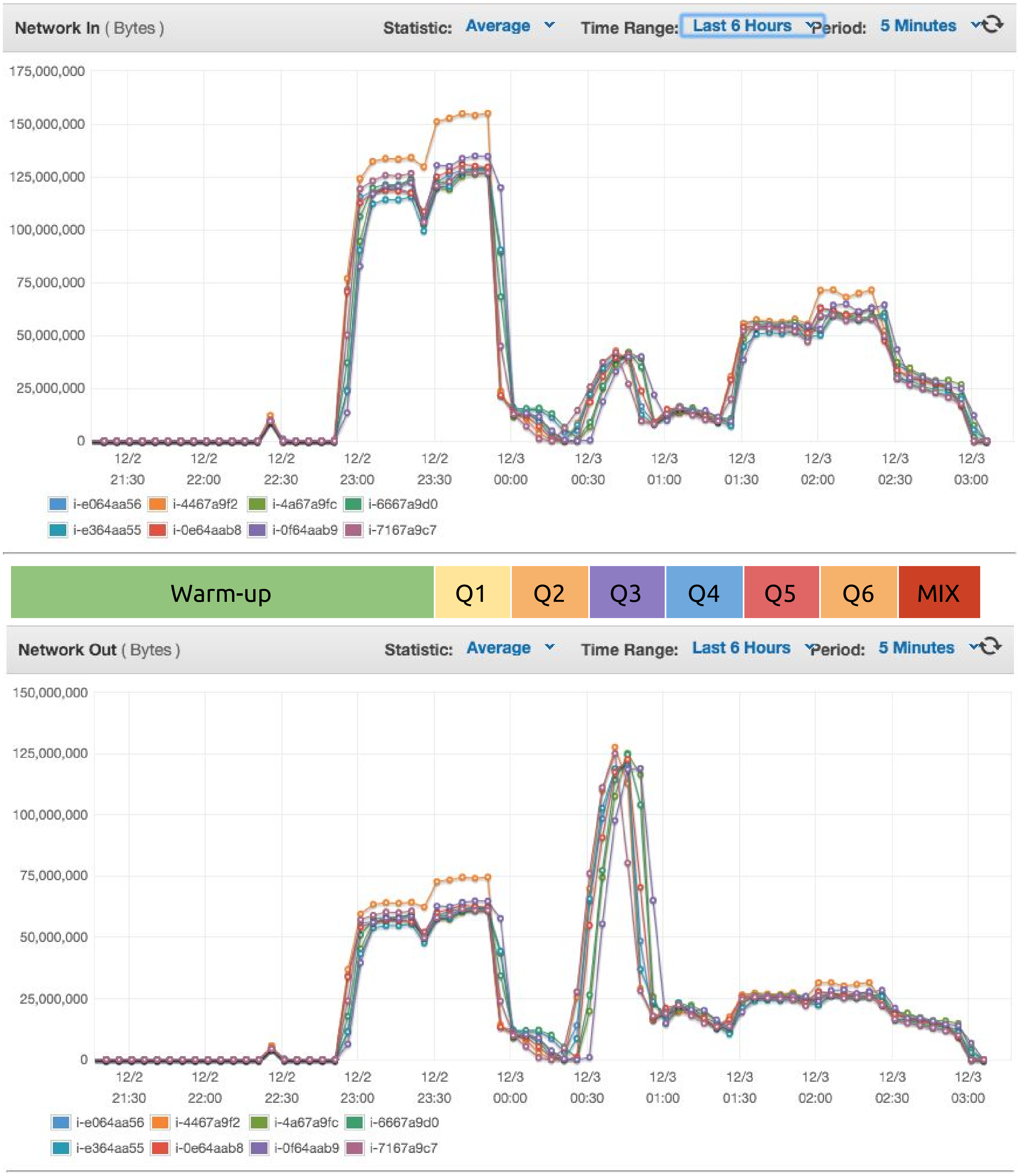
Period: 1 Minute


Healthy Hosts (Count)
Statistic: Average
Time Range: Last 6 Hours
Period: 1 Minute
9
8
7
6
5
4
3
2
1
0
12/2 21:30
12/2 22:00
12/2 22:30
12/2 23:00
12/2 23:30
12/3 00:00
12/3 00:30
12/3 01:00
12/3 01:30
12/3 02:00
12/3 02:30
12/3 03:00
a
Warm-up
Q1
Q2
Q3
Q4
Q5
Q6
MIX
Average Latency (Milliseconds)
Statistic: Average
Time Range: Last 6 Hours
Period: 1 Minute
140
120
100
80
60
40
20
0
12/2 21:30
12/2 22:00
12/2 22:30
12/2 23:00
12/2 23:30
12/3 00:00
12/3 00:30
12/3 01:00
12/3 01:30
12/3 02:00
12/3 02:30
12/3 03:00
a



Sum Requests (Count) Statistic: Sum Time Range: Last 6 Hours Period: 1 Minute

CPU Utilization (Percent) Statistic: Average Time Range: Last 6 Hours Period: 5 Minutes



Network In (Bytes)

Statistic: Average

Time Range: Last 6 Hours

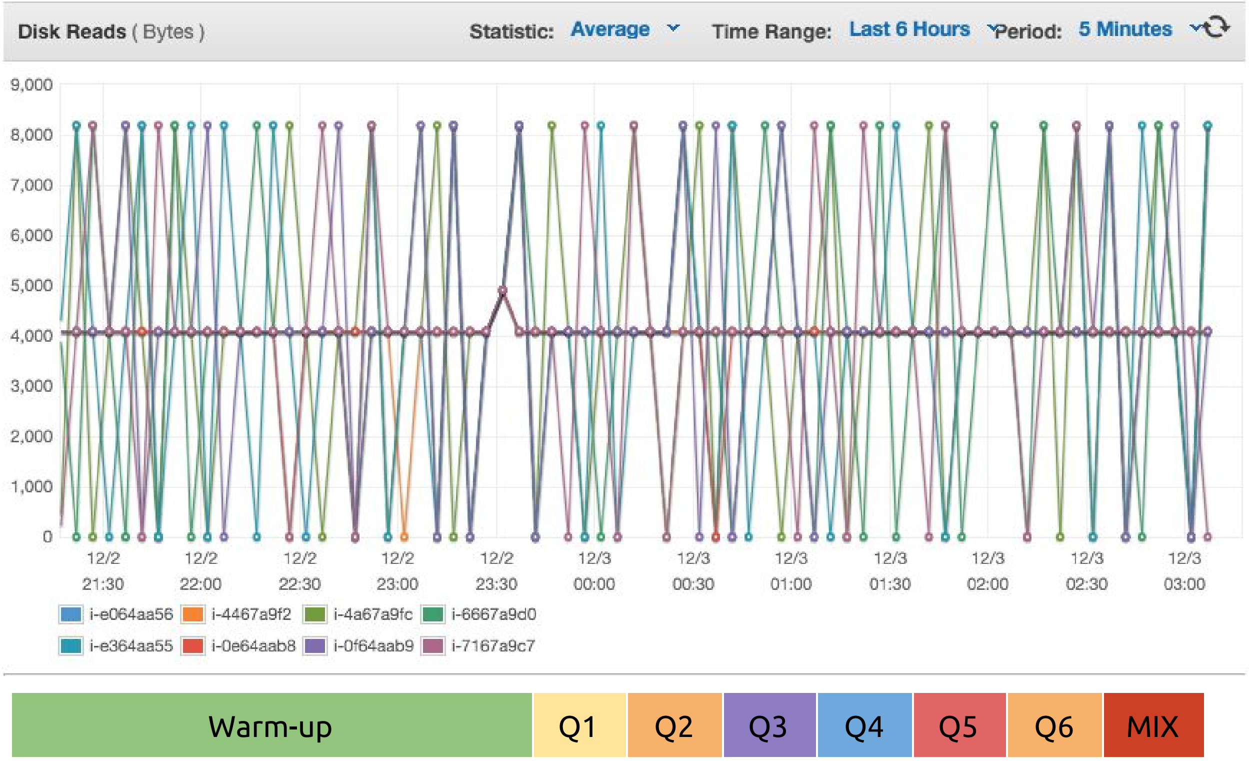
Period: 5 Minutes

Network Out (Bytes)

Statistic: Average

Time Range: Last 6 Hours

Period: 5 Minutes



9. What was the cost to develop the front end system?

As we put frontend and backend database on the same instance, we cannot calculate the cost of frontend and backend separately. However, we can calculate the total cost of instances without counting EBS cost. The total cost is $0.133/h * 8 (on demand cost of m3.large instances) + $0.025/h (ELB) = $1.089/h

10. What are the best reference URLs (or books) that you found for your front-end?
https://hbase.apache.org/apidocs/org/apache/hadoop/hbase/client/package-summary.html
http://zetcode.com/db/mysqljava/
http://vertx.io/docs/vertx-core/java/

11. Do you regret your front-end choice? If yes, what would you change in the way you approached Q1?

No.

[Please submit the code for the frontend in your 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.

   Q1 We are not using any database.

   Q2 We use MySQL database. We combine userid and tweet time as one single column to be the index, and we only need to look up into the database once. Before this, we have tried using composite index on userid and timestamp as two separate columns, and we thought combine them into one will same lookup time. If time permits, we will try to modify the index type from char to bigint, which will speed up searching time.

   Q3 We tried storing each tweet as a single line and this will inevitably cause sorting and range search in database, which will cost much time and give low rps. So we decided to change the schema this time.
   We have one user have only one line in database, containing all tweets information of this user. So when we want to get information about this user, we do one accurate search in database, and we create index on userid. We store all data of that user in json format, and read the information in frontend to do filtering and sorting. This makes the query speed up a lot.

   Q4 We follow the schema we use last time. That is, we process all data in ETL process. We use hashtag + date as key, and the earliest tweet in that day, all users in with that hashtag in that day all in the content. We then sort the table based on the times of hashtag being tweeted that day. In this way, when we try to get the days that publish most tweets with that hashtag, we can easily do 'limit' operation so we can avoid using any kind of sorting or range search.

   Q5 We first tried storing userid-tweet count in one line and sum up all tweet count in the range of users. However, when the range is big, it took 10min to finish querying. This will lead to read time out. So we tried dynamic programming to have each user id correspond to the count of tweets of all users with userid smaller than this person. However, the querying id may not exist, so we need to have two arrays to store the userid and count and do binary search to find the corresponding total count. As the data is relatively small to 0.6GB, we read file to the server and the initialization stage so we can do this in memory when querying.

	Backend Server	Schema
Q2	MySQL	CREATE TABLE `q2` (`uidtime` CHAR(36) NOT NULL, `text` TEXT NOT NULL, `tweet_id` BIGINT NOT NULL, `score` int NOT NULL) ENGINE=MyISAM DEFAULT CHARSET=utf8mb4;

Q3	MySQL	CREATE TABLE `q3` (
		`user_id` BIGINT NOT NULL,
		`content` LONGTEXT NOT NULL
		) ENGINE=MyISAM DEFAULT CHARSET=utf8mb4;

Q4	MySQL	CREATE TABLE `tweetsq4` (`tag` VARCHAR(2048) NOT NULL , `date` DATE NOT NULL, `count` INT NOT NULL, `users` LONGTEXT NOT NULL, `tweet_id` BIGINT NOT NULL, `text` TEXT NOT NULL) ENGINE=MyISAM DEFAULT CHARSET=utf8mb4; create table q4 as(select * from tweetsq4 order by count desc, date asc);
Q5	Memory	
Q6	MySQL	

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

   In our design as described above, we try to minimize backend operations to guarantee high throughput, for Q2, Q4 the queries are simple lookups, and for Q3 the queries become range queries plus operations (sorting). So we created hash indexes on Q2 and Q4. Without index, for random query, the database will have to scan through the table for the desired result. For Q3, we solve this problem by have only one line of user and its tweet content and manage all the other logic like sorting and ranging in the frontend.

   The measures we adopted:
   MySQL

Q2: ALTER TABLE q2 ADD INDEX IDX_UID(uidtime) USING HASH;
Q3: ALTER TABLE q3 ADD INDEX IDX_UID(user_id) USING HASH;
Q4: ALTER TABLE q4 ADD INDEX IDX_HASHTAG(tag) USING HASH;

HBase

In our rowkey design for HBase, our records are ordered in the way similar to the MySQL indexes for each queries.

3. Explain (briefly) **the theory** behind (at least) 10 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).

   common

   a. creating index (index in MySQL and rowkey in HBase)
   b. increasing concurrency by using more threads
   c. limit maintenance tasks during service time
   d. use a connection pool and long lived tcp connection
   e. use a fast underlying storage media, like flash with provisioned IO

   MySQL

   f. increasing buffer size for caching table
   g. increasing buffer size for caching table index
   h. increasing buffer size for caching query result
   i. query optimization, give hints about the underlying tables, to allow query planner to better make plans

   HBase

   j. read shortcircuit of HBase and HDFS
   k. increasing jvm heap memory size

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).

   Below are the relative score for all queries throughout the entire team project. We have changed our design several times in database table design and feature, and overall system architecture design.

   For Query 1 we achieved good performance even on single node, however we meet difficulty of scaling out with ELB, with ELB we didn’t achieve expected throughput gain. There is theory that because the frontend server is taking much less time than the ELB scheduling, the throughput drops. Though not fully convinced, we tried using a single instance behind ELB. There is another theory is that the load generator isn’t going too far beyond the maximum request.

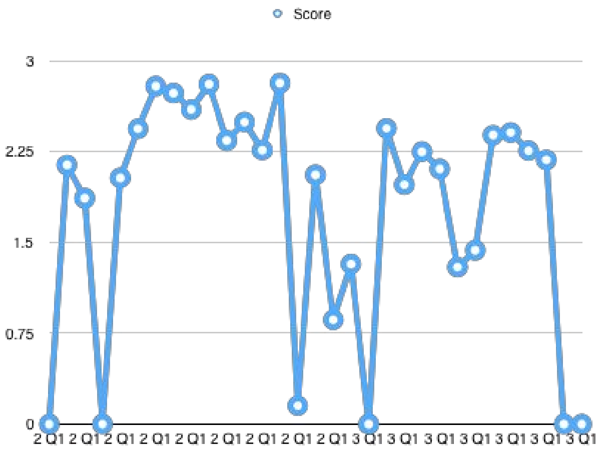
For Query 2, Query 3, and Query 4, we preprocess the data so that each query would result in only one database lookup.

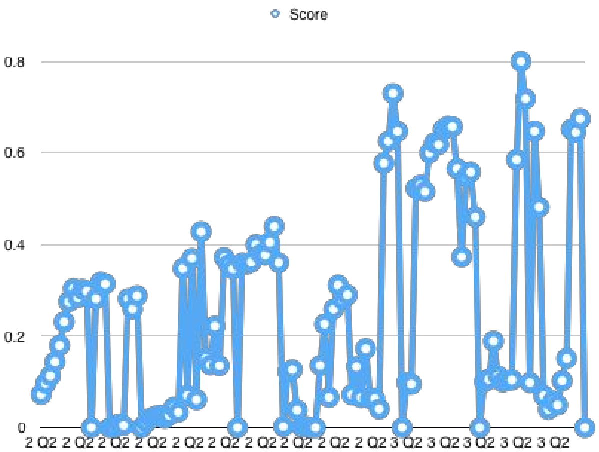
For Query 5, we preprocessed the userid and tweet counts, so that range query will only take two lookups and subtract the two results gives the requested result.

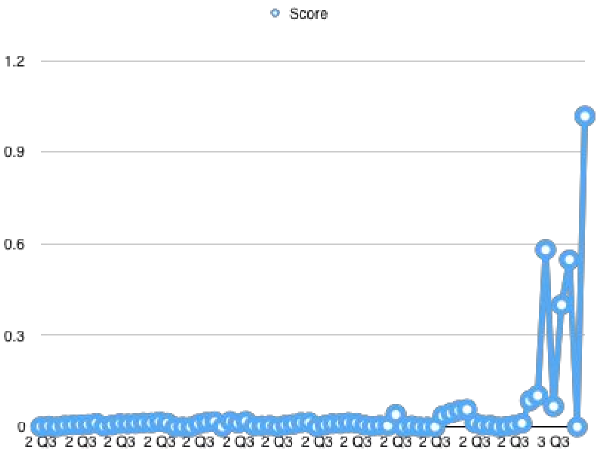
For Query 6, we save the transactions at frontend server, and request for the censored text from Q2 database.

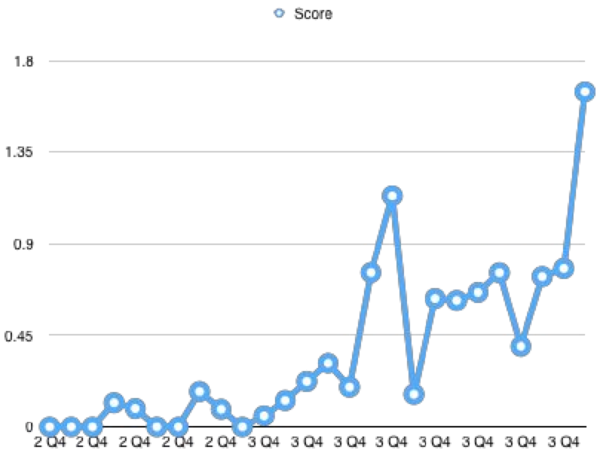
Some other techniques we tried are:

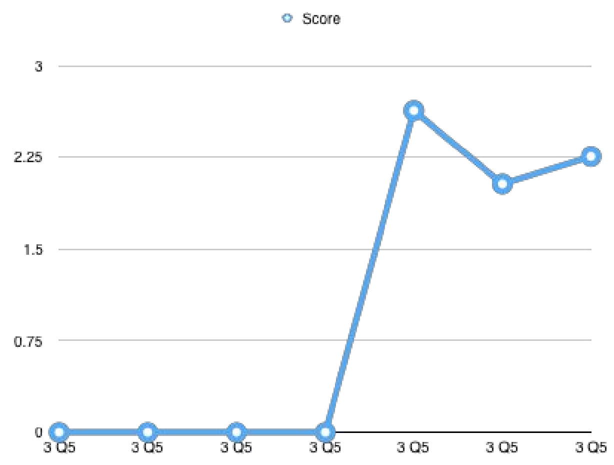
1. all the database optimizations as described above
2. use multiple frontend to access only local MySQL database
3. use single frontend to access MySQL of many backend server











5. How did you implement writes? Did you change your design in order to handle put requests? Why or why not?

   For Q2 and Q3 it will work, but for Q4 it won't work. Because for the performance of getting the result, the preprocessed result is stored so the table is not ready for insert/update. For Q5, it is a statistics query, and we preprocessed the result for faster lookup, so if there's any update, the lookup table need to be rebuilt.

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

   Standard ways of connection to the database: JDBC, HBase client

7. How did you profile the backend? If not, why not? Given a typical request-response for each query (q1-q6) 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)
   f. DB to Web Service
   g. Parsing DB response
   h. Web Service to LB
   i. LB to LG

   How did you measure this? A 9x6 table is one possible representation.

   For MySQL, we profile single SQL statements during development, and use the admin tool in tests.

A rough estimate of the latency:

Item	Q1	Q2	Q3	Q4	Q5	Q6
Load Generator to Load Balancer	1ms	1ms	1ms	1ms	1ms	1ms
Load Balancer to Web Service	1ms	1ms	1ms	1ms	1ms	1ms
Parsing request	1ms	1ms	1ms	1ms	1ms	1ms
Web Service to DB	~0ms	1ms	1ms	1ms	~0ms	~0ms
At DB (execution)	~0ms	110ms	7ms	7ms	1ms	~0ms
DB to Web Service	~0ms	~0ms	~0ms	~0ms	~0ms	~0ms
Parsing DB response	~0ms	3ms	8ms	8ms	~0ms	~0ms
Web Service to LB	1ms	1ms	1ms	1ms	1ms	~0ms
LB to LG	1ms	1ms	1ms	1ms	1ms	~0ms
Total	5ms	117ms	19ms	19ms	5ms	2ms

Item	Q1	Q2	Q3	Q4	Q5	Q6
Load Generator to Load Balancer	20.00%	0.43%	2.63%	2.63%	11.11%	33.33%
Load Balancer to Web Service	20.00%	0.43%	2.63%	2.63%	11.11%	33.33%
Parsing request	20.00%	0.43%	2.63%	2.63%	11.11%	33.33%
Web Service to DB	0.00%	0.43%	2.63%	2.63%	0.00%	0.00%
At DB (execution)	0.00%	94.02%	36.84%	36.84%	22.22%	0.00%
DB to Web Service	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%
Parsing DB response	0.00%	2.56%	42.11%	42.11%	0.00%	0.00%
Web Service to LB	20.00%	0.85%	5.26%	5.26%	22.22%	0.00%
LB to LG	20.00%	0.85%	5.26%	5.26%	22.22%	0.00%

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).

   Google Map and Google Earth uses BigTable[1] (NoSQL) to store the image blocks

[1] Chang, Fay, et al. "Bigtable: A distributed storage system for structured data."*ACM Transactions on Computer Systems (TOCS)* 26.2 (2008): 4.

because it's scalability, availability and fault tolerance. Google would use RDBMS database for its advertising services. [2]

9. What was the cost to develop your back end system?
   20 hours of
   EC2 8 x m3.large => $5.71
   EBS 120 GB x 8 instances => $4.13
   ELB 187.567 GB => $1.22
   Total: $11.06

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

    [MySQL :: MySQL 5.7 Reference Manual](https://dev.mysql.com/doc/refman/5.7/)
    [Chapter 14. Apache HBase (TM) Operational Management](http://hbase.apache.org/0.94/book/ops_mgt.html#import)
    [An In-Depth Look at the HBase Architecture | MapR](https://www.mapr.com/blog/in-depth-look-hbase-architecture)

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

[2] Shute, Jeff, et al. "F1: the fault-tolerant distributed RDBMS supporting google's ad business." *Proceedings of the 2012 ACM SIGMOD International Conference on Management of Data*. ACM, 2012.

Task 3: ETL

1. For each query, write about:
    a. The programming model used for the ETL job and justification

        MapReduce. MapReduce is useful when dealing with big datasets.

        For Q2, we print our columns we want in the mapper period, and filter tweet and calculate sentiment score in the reducer period, and add these two columns to the output.

        For Q3, mapper extract columns we need which are user id, tweet id, tweet time text and follower count. In reducer, we censored text and calculate total score based on sentiment score and follower count. We score the output in json format and then transform into tsv later.

        To optimize, we only print out tweets with sentiment score not equal to zero, so this will give us less results but still cover all what we want. Also, we have tweets that have sentiment score greater than zero in on file, and tweets with sentiment score below zero in one file so later we can have two tables.

        But we found out that this does not provide good performance, so we decided to change the schema, to have all tweet data of one user in one line. This will reduce time in searching in database, but rather put more logic on front end. To do this, we rerun the data from last time using EMR and combine all tweets of the same user in reduce period and save them in json format.

        For Q4, in ETL, we process data to the final format we want queried by front end. That is, each hashtag and date correspond to one count of hashtag in that day and all users, also the first text in the day.

        In mapper, we use hashtag+date as key, so the tweet with the same hashtag on the same day will be sent to the same reducer. In reducer, we combine all tweets on the same day and only leave the one with the earliest time.

        To save like this, instead of each tweet in one record, we can save time when we query, because we get the result back to the front end server directly.

        For Q5, in map period, we simply print out user id and tweet id, so in reducer, tweets from the same user will be sent to the same reducer and close to each other, so we can count the number of distinct tweets for each user.

    b. The type of instances used and justification

c3.8xlarge for Q5
Instances in C group compute fast and 8xlarge has more cores that will speed up the data processing period. Although one instance costs much, we save time. So in total, we get about the same amount of money as we use xlarge instances or 2xlarge instances, but we save time, which is a big deal.

c3.xlarge for refining Q3 data.
As Q3 data has already been cleaned so we only need deal with 15GB data, and the logic is also pretty easy to combine all all tweets of one person together. So using c3.xlarge is enough.

c3.xlarge for refining Q2 data.
The logic for changing Q2 data schema is really easy to concatenate userid and time together so c3.xlarge is enough.

c. The number of instances used and justification

   1 master + 8 cores for Q5
   More cores will help compute faster and save time while spending about the same amount of money when using 4 cores to wait for 4h or 8 cores to wait for 2h.

   1master + 4 cores for Q2, Q3
   1+4 is enough to keep each run in one hour, which can save money.

d. The spot cost for all instances used

   For Q5, we use spot price $1 for each c3.8xlarge instance. And in reality, the actual price is around $0.44. The overall cost is ($0.27 per hour for EMR c3.8xlarge + $0.44 per hour for one c3.8xlarge instance) * 9 * 2h = $12.78

   For Q2 and Q3, each is ($0.035 per hour per c3.xlarge instance + $0.053 per hour per EMR c3.xlarge)*5*1h = $0.44

   Total = 12.78 + 0.44*2 = 13.66

e. The execution time for the entire ETL process

   We didn't rerun data for q2-q4. For q5 alone, it took 2 hours to run EMR job. We can suppose we spend the same amount of time cleaning data for q2-q4. After cleaning data and get tsv files, we need to load it into databases.
   It will take 30min to load q2, 1h to create index; 10min to load q3, 30min to

create index; 10min to load q4, 20 min to create index.
When we get a brand new instance and all raw data, and want to have all the data ready in the MySQL database, it will take 2h*4 + 30min+1h+10min+30min+10min+20min = 10h40min

f. The overall cost of the ETL process

$12.78

g. The number of incomplete ETL runs before your final run

0. We succeeded in one run for q5 and we didn’t rerun for q2-q4.

h. Discuss difficulties encountered

For refining Q3, we want to have json format data stored in database, so we can use java to read json directly into frontend and deal with logic in frontend. However, MySQL will treat \\n as \n when load data, \\” as \”. I’ve tried many times to make what is in the database in the format of Java json format. Also, I use python to process data and Java to load data from database, there is a transformation problem. Finally I solved it by have python json data and replace all \ by \\ and load this into MySQL and succeeded.

i. The size of the resulting database and reasoning

Q2: 30G data + 10G index
Q3: 15G data + 5G index
Q4: 4G data + 1G index

We didn’t use database for Q1 and Q5.
total: 74G

j. The time required to backup the database

We didn’t backup the database. Instead, we save the input tsv file on s3 and scp to the instance every time we have a new instance and load data into database. As our data loading time is relatively fast, we can do this.

k. The size of the backup

Q2 30G + Q3 15G + Q4 3.7G + Q5 0.6G = 49.3G

2. Critique your ETL techniques. Based on your experiences over the past 6 weeks, how

would you advise a student who attempts this project next semester?

1) Be aware of escape characters
2) Be aware of unicode characters
3) Talk to your teammates to make sure what kind of data you really want.
4) Design your schema carefully before you start running EMR jobs.
5) Budget is limited.
6) But don't save money by have very small type of instances and a small number of instances. You will end up taking days to finish EMR job.
7) Test on small data be for run on EMR.
8) Ask AWS to increase the limit of spot instances at the very beginning of your project.
9) Try to print out tsv format instead of json format, so you don't need to transform later and tsv file is more friendly when loading into database.
10) Be aware that Mysql will take input \\n as \n when loading data.

3. What are the most effective ways to speed up ETL?

1) In extracting period, use larger type of instances and more instances.
2) In transform period, think carefully what kind of schema to use before running EMR, so we can do transformation during EMR. If we have data cleaned once, and want to change schema in the backend, then we will need to use EMR again, which is really a waste of time and money.
3) In loading period, we tried MyISAM engine instead of InnoDB, and it speeds up the loading progress. Also, don't indicate primary key or indexed when creating tables in Mysql. Add Indexes after all data has been successfully loaded.

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

Yes. We used streaming. We didn't try non-streaming EMR.

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

No. We use 'LOAD DATA INFILE' function to load Mysql databases.

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

We didn't use HBase in this phase. We use all Mysql as backend database for query q1-q6. It is easy to load to Mysql, as Mysql provides good internal loading tool, which is easier to use and also very fast. It took us less than 10 mins to load data for q3 and q4 when data is relatively small, and around 20min-30min to load 30G data for q2. From previous experience, HBase is also easy loading data, we can use 'bulk load' tools, but we need to concatenate all files into one, because this method performs best when loading the first trunk of data into database.

In all, I would say MySQL is easier to load.

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

General Questions

1. What are the advantages and disadvantages of MySQL for each of the queries you've encountered so far? Which queries are better suited for MySQL (not HBase)?
   MySQL is more stable but do not have a good property in scaling. For queries that are doing accurate search like q2, and q4, using MySQL by creating index in good choice.

2. What are the advantages and disadvantages of HBase for each of the queries you've encountered so far? Which queries are better suited for HBase (not MySQL)?
   HBase is good at scaling but not as stable as MySQL. For queries like Q3 before modification, we are using range search in MySQL, but by creating appropriate row-key, HBase can perform better in range search by limiting the range indicated by row-key. Also, when we are unsure about the length of data in one column, HBase is also a good choice.

3. For your backend design, what did you change from Phase 1 and Phase 2 and why? If nothing, why not?
   In previous phases, we store the the data in the database in the most convenient way as the data is, and write functions in database (MySQL) to process the query and have the result ready for the frontend server to reply to the client. But for HBase, there's no support for this, and we fit all data in one HBase column for query, and this proved to be very efficient, so in this phase we preprocessed the data so that all related information needed for Q3 and Q4 is stored in a single database, and let the front-end server process the result according to the request. We decided to design it this way because in a high throughput system, if database executes complex logic, then it's very likely that it will become the bottleneck of the system because different queries will competing for disk IO resources.

4. 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 heavily depend on MySQL, and each frontend server is deployed with a MySQL server with complete dataset. This doesn't scale, so it might handle if the data would double, but not 10 times larger. It's because the cost for storing the data multiple times is really high, from the cost structure of our project we see that we are spending half the budget on storage, and half on computation.

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

   We used Apache Benchmark to generate load for warming up ELB.

6. How did incorporating writes into your system cause you to undergo a redesign?

For Query 6, the update doesn't need to be persisted, so it's not much different but to store the states on the server and discard afterwards. If updates were to be persisted, then most of our preprocessing won't work. Depending on the update frequency and consistency requirement, we can choose NoSQL, SQL database or a combination of both to process transaction and analytics requests.

7. How would you have dealt with writes to one of the other queries (like Q3, Q4) where each read was a range query?
   We should design a pipeline that take in the transaction, and updates also feed the update to our analytics database so that queries can be executed fast. It also depends on the consistency requirement for updates and accuracy requirement for queries.

More Questions (unscored)

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

9. What were the five coolest things you learned in this project?

10. Which was/were the toughest roadblock(s) faced? What was the solution to that problem?

11. Design one interesting query for next semester's students.

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

13. How will you describe this project (in one paragraph) on your LinkedIn / CV ?