15619 Project Phase 3 Report

Performance Data and Configurations

Live Test Configuration and	Results
Instance type	m1.large
Number of instances	6
Cost per hour	$1.134
Queries Per Second (QPS)	q1/q2/q3/q4/q5/q6/MIX[q1/q2/q3/q4/q5/q6/] INSERT HERE: score[98/19/71/71/67/128/MIX[242/31/50/30/57/235]] tput [13655.2/1010.8/7219.7/3176.1/4990.1/9616.4/MIX[7261.0/584.8/969.7/579.8/1072.6/4222.1]] latcy [7/49/6/14/9/5/MIX[6/42/25/42/23/5]] corr [100.00/96.00/99.00/100.00/100.00/100.00/MIX[100.00/94.00/92.00/93.00/95.00/100.00]] error [0.00/0.12/0.00/0.00/0.00/0.00/MIX[0.00/0.10/0.06/0.10/0.03/0.00]]
Relative Rank [1 - 91] :	Phase 1 : 47 Phase 2 : 5 Phase 3 : 36
Phase Score [out of 100]	====== graded ====== Phase 1: 24 Phase 2 Live (selected, mention DB): 97 MySQL Phase 3 Live: 67 ====== others ====== Phase 2 (Pre-Live): 47 Phase 2 Live (dropped, mention DB): 0(not using HBase in this test) HBase Phase 3 (Pre-Live): 44

Team : sudoCloud

Members : Qinyu Tong <qtong@andrew.cmu.edu>,

Huacong Cai <hcai@andrew.cmu.edu>,

Hansi Mou <hmou@andrew.cmu.edu>,

[Please provide an insightful, data-driven, colorful, chart/table-filled, humorous and interesting final report. This is worth a quarter 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.

   We use **Wildfly** as our web application server, and it uses **Undertow** as its web server.

   We choose this framework due to our study of benchmark from Techempower, which shows that undertow perform well in different kind of benchmark in EC2. And Wildfly is an open-source, mature, flexible, lightweight application server which use undertow as its web server.

Properties	Details
1.Unparalleled Speed	a. Fast Startup b. Ultimate Web Performance & Scalability
2.Exceptionally Lightweight	a.Memory Diet b.Slimable/Customizable Runtime
3.Powerful Administration	a.Unified configuration & Management b.Domain & Standalone Management
4.Supports Latest Standards and Technology	a.Java EE 7 b.Modern Web
5.Modular Java	a.No more jar hell! b.Fast Linking & Concurrent Loading
6.Smarter Development	a.Arquillian b.Smarter Development
7.Based on the Best of Open Source	Hibernate Narayana Infinispan IronJacamar RESTEasy Weld HornetQ JGroups Mojarra Apache CXF Arquillian

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

   m1.large is the most powerful instance type that we can choose in EMR (CPU, Memory and Network), thus we use this instance type.

In order to handle the heavy workload, we need more instances to handle requests. But we are constrained by the budget. After careful calculate the budget and run tests to observe the performance, we finally decide to use 6 instances. 1 Master node and 5 Core nodes, and deploy front end on every one of them.

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

   First of all, we configure the max task threads of Wildfly to handle more requests at the same time.

   Secondly, we increase the initial memory allocation pool (Xms) and the maximum memory allocation pool (Xms) for our the Java Virtual Machine (JVM) that Wildfly uses. Both of them are set to 2560 MB (2.5 GB).

   Thirdly, because there is not a good solution for Q6, i.e. we cannot get a great performance if the query id is not in the database, we cache the data of Q6 in our front end system.

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

   We use HBase as our backend, this is because that we specified to grade on MySQL in our phase 2. And due to that, the APIs we use to communicate with backend system have to change.

   And we don't cache any result in our code any more, because we have do many front end instance, the hit ratio of Q1 is low, and complex code will have more overhead. The other queries have a large set of results, simply cache them will lead to leak of memory, but implement complex cache mechanism (automatically clear the timeout contents or earliest contents when cache is full) will have too much overhead.

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 we use ELB for our front end system. Because it is really a heavy load on the live test. Single front end system will have limited computation, memory and network capacity, which cannot fulfill the requests and thus results in bad performance.

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.

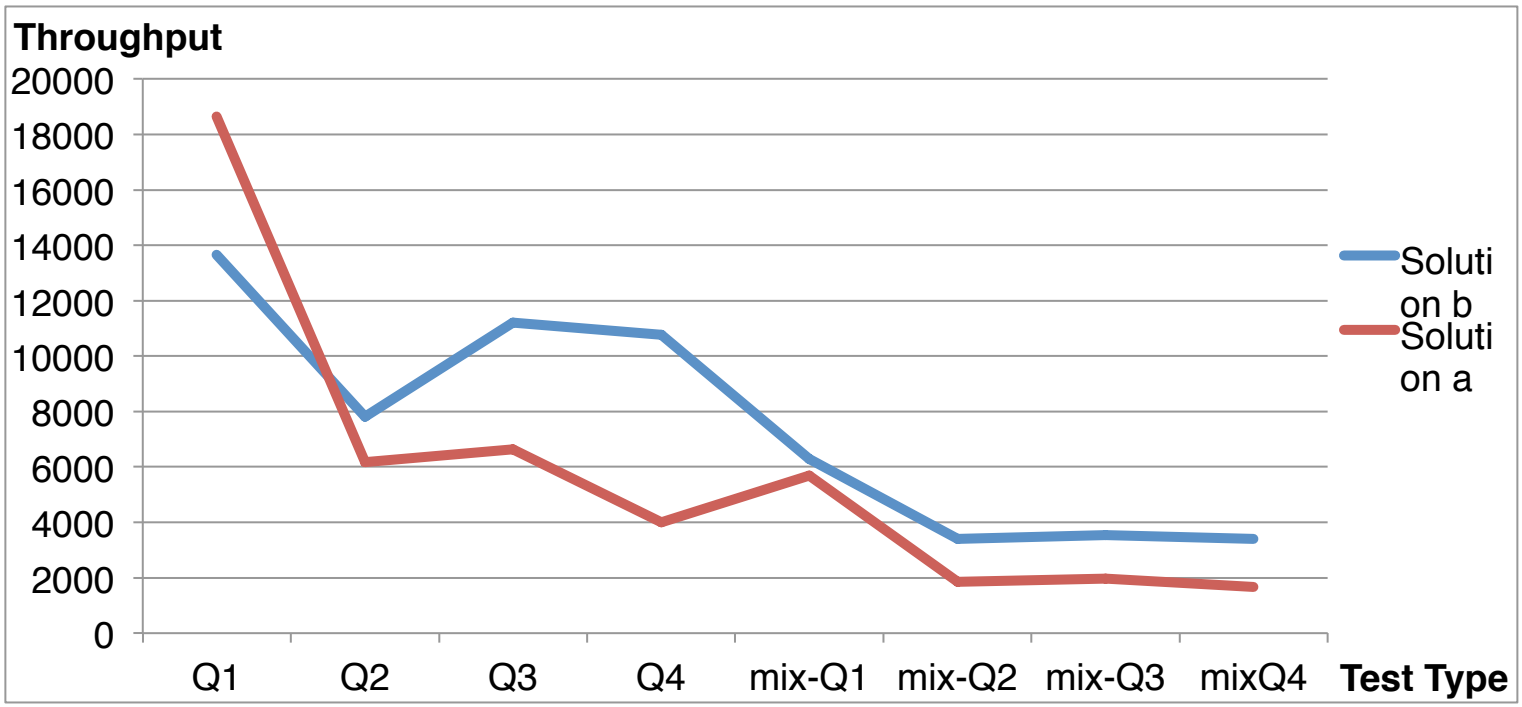
Yes, we have tried:

a. Single front end system, using ELB to connect to multiple back end systems (MySQL as backend)

b. Front end system integrated with back end system (Fully independent back end system when using MySQL, i.e. each one has all data. Or on both master node and core nodes when using HBase). And using ELB to connect to these front end instances.

Another solution we haven't tried is using ELB to connect to multiple front end system, and all the front end system using ELB to connect to back end system (MySQL or multiple HBase cluster) or directly connect to the same back end system (One HBase cluster). Because we are limited by the budgets, we cannot have enough budgets for so many instance. And too much ELB layer will increase latency, and deploy front end with back end on the same instance will reduce the time for transfer data (all data in MySQL, or some data in HBase if the target region server is the same as the front end system)

We finally decide to use solution b, because multiple front end can handle more request, and won't be limited by single instance's computation, memory and network capacity.



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

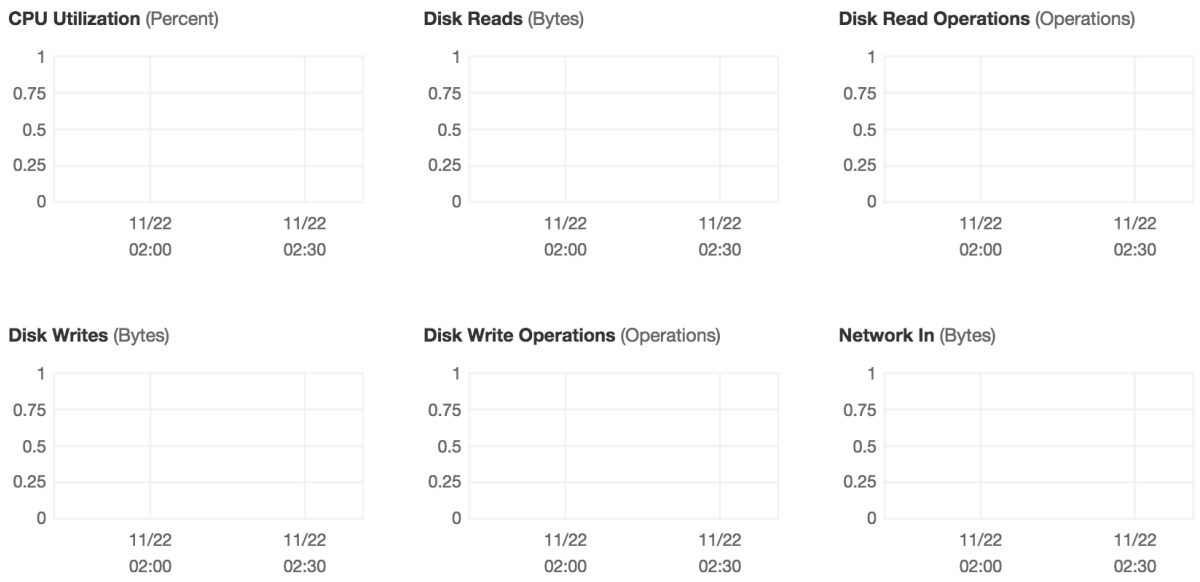
Yes, when using MySQL as back end database, the configuration we make including install jdk 1.8, configure environment path, install Wildfly, configure Wildfly, redirect traffic

from port 80 to 8080 (because we don't run Wildfly with root privilege, thus it can only listen to port 8080 instead of known HTTP port 80), deploy our application, etc. When

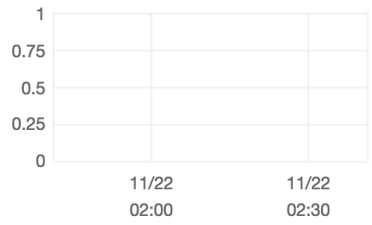
8. Did you use any form of monitoring on your front-end? Why or why not? If you did, show us the results.

   Of course we monitored our front-end system, because we need to track the status of our front end system and adjust it according to the data we get. The tools we used for monitoring including CloudWatch for instances and ELB provided by Amazon, and the management system provided by Wildfly.

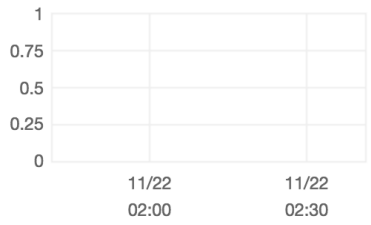
   Graph of CloudWatch:



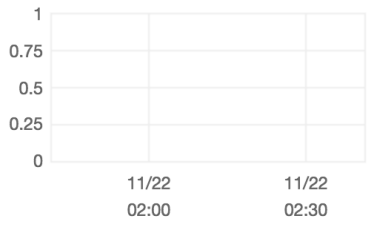
Network Out (Bytes)



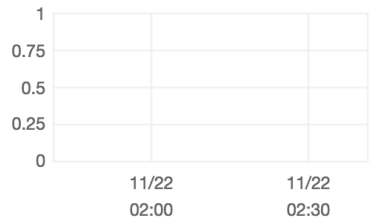
Status Check Failed (Any) (Count)



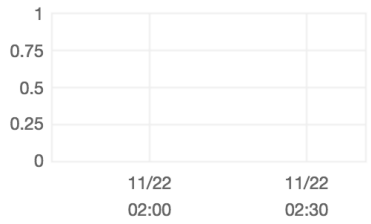
Status Check Failed (Instance) (Count)



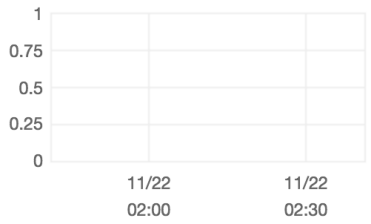
Status Check Failed (System) (Count)

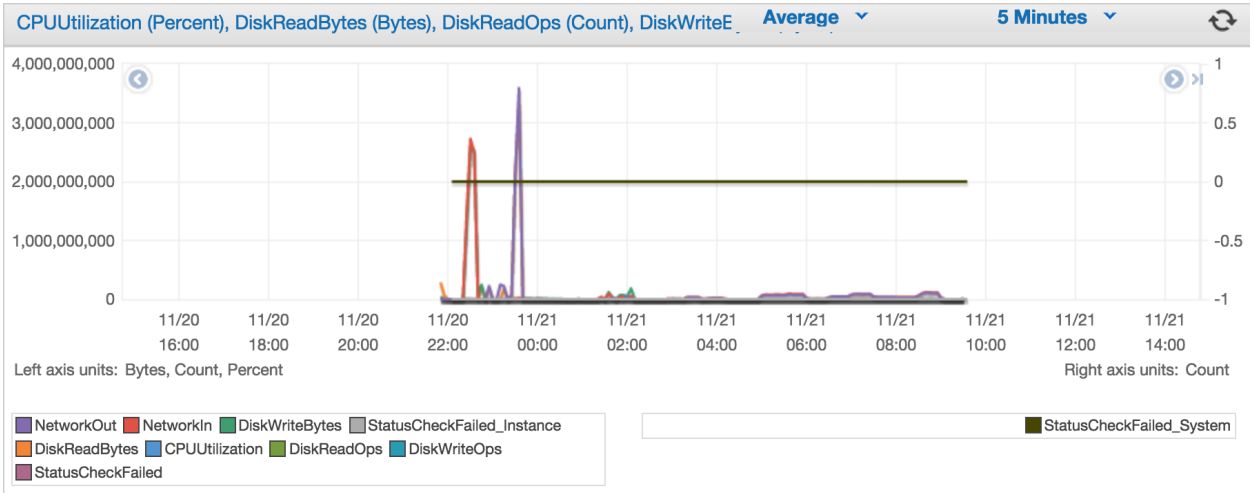


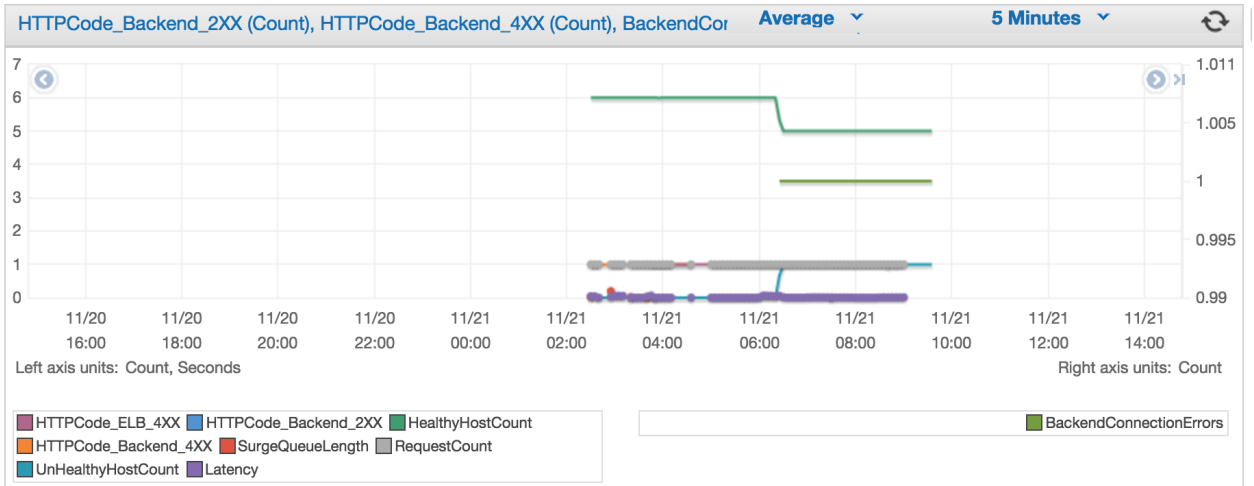
CPU Credit Usage (Count)



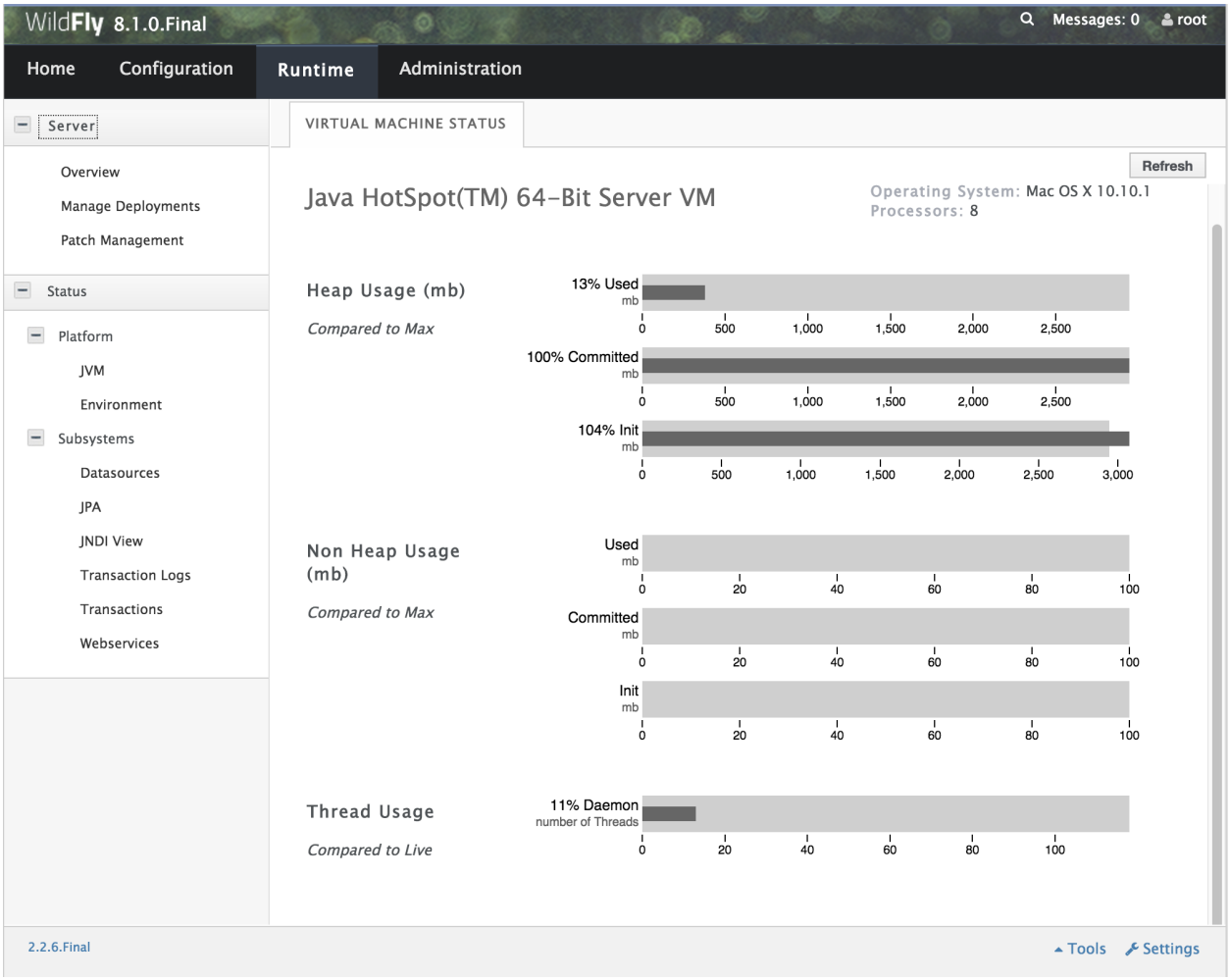
CPU Credit Balance (Count)







Graph of Wildfly management system:



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

   We use the same cluster as our back end system in phase 3, and when developing the front end system, the cluster is also used for import data or doing something for developing back end system.

   The total cost (including front end and back end) is $8.

10. What are the best reference URLs (or books) that you found for your front-end?

    Wildfly:
    Official documentation:
    https://docs.jboss.org/author/display/WFLY8/Documentation
    Official github:
    https://github.com/wildfly/wildfly

Servlet:
Head First Servlets & JSPs (O'Reilly)

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

    No, our front end framework is open-source, mature, flexible, lightweight, and the most important features are reliable and fast. But if you mean how to beat Q1, we might pre compute and cache some results to avoid some real time computation, though we don't have the chance to test this idea.

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.

   Our schema design principle is that keeping the query and response as simple as possible to minimize the front end's processing delay. In the ideal case, the front just queries and then forwards the response without any parsing. So in hbase, we use query as rowkey and response as column family for all Qs.

   HBase schema design:

   q2 table:

Row Key	Column family: response
user_id+time	tweet_id:score:tweet

   q3 table:

Row Key	Column family: response
user_id	retweet_user_ids

   q4 table:

Row Key	Column family: response
date+location+rank	hashtag:tweet_ids

   q5 table:

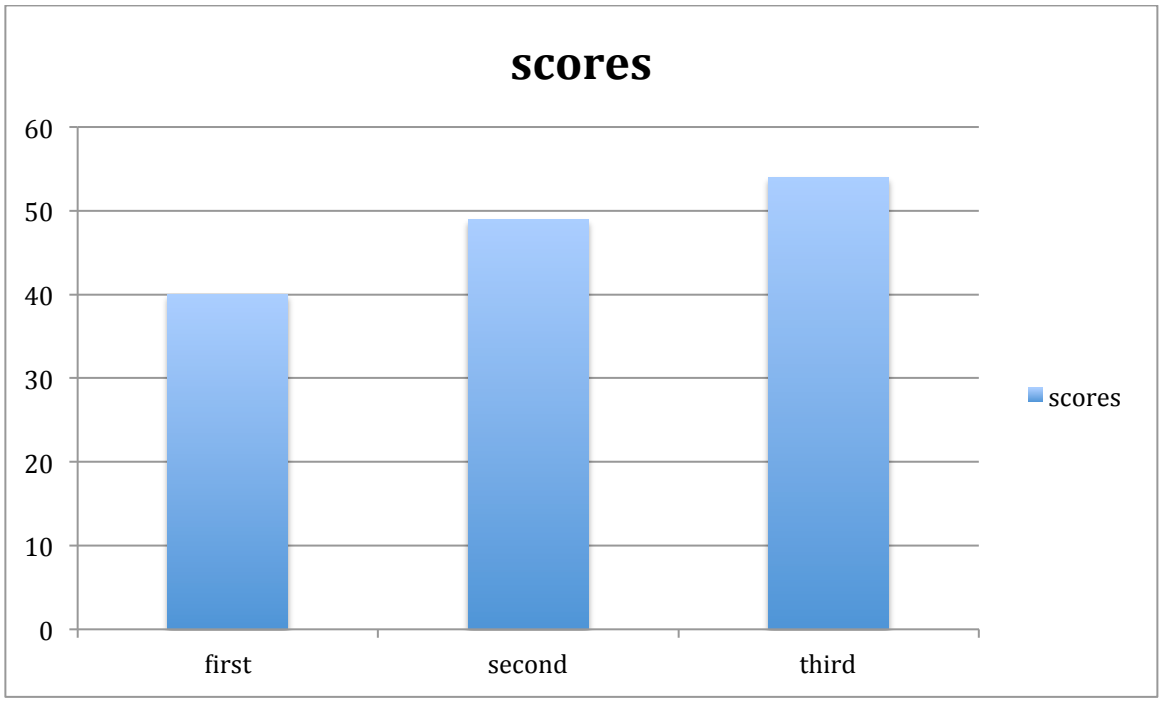
Row Key	Column family: response
user_id	score1+score2+score3+total

   q6 table:

q6 table is a little trick, we do not simply store the user and user's photo number. Because the query ask the sum of photos of user m to n, so we compute the sum of former x users' photos in ETL. And what we need to do in the front end, is just selecting two entries and differencing the sum.

Row Key	Column family: response
user_id	sum of photos of all users until this user

We have three irritations of our schema design:
In version 1.0, our column family has many columns. For example, in q2, a column family has tweet_id, score, censored_text, three columns. Front end need to combine three columns to form the response.
In version 2.0, we combine those columns into a single unit, thus a table becomes like query-response format.
In version 2.0, we enable the bloom filter of the table, this enhance the cache hit ratio.



If we do not using hbase as the database for backend, our design schema in MySQL is like following scenario:
MySQL:

q2 table: Index (tweet_time, user_id)

user_id	tweet_time	response (tweet_id:score:tweet)
bigint	datetime	mediumtext

Using bigint as user_id's type can reduce the store size of user_id and can also provide faster look up than varchar type since int compare are faster than string compare. The reason for using datetime for tweet_time is the same as user_id.

As for the index, we place tweet_time at the first place because we compare the response time with the index (user_id, tweet_time) and found that index(tweet_time, user_id) is faster than index (user_id, tweet_time). The reason, in my opinion, is that comparing datetime is faster than comparing bigint. Another reason is that the cardinality of date in is less than user_id. Therefore, the first step search(match a given date) takes less time than match a given user_id.

q3 table: index (user_id)

user_id	retweet_user_ids
bigint	mediumtext

q4 table: index (date,location,rank)

date	location	rank	response (hashtag:tweet_ids)
date	varchar	int	mediumtext

By store the rank in database, the query that front end submit will be like "select response from q4 where date=? and location=? and rank>=m and rank <=n". Front end can return the results immediately without further processing.

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.

   Hbase's hot region problem, when the query pouring to the same region server, this will result a high delay. We plan to solve it by add hash value to the Rowkey to split entries evenly to all regions.

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).

   3.1 Import data with a table without index. When we import data to MySQL with table indexed, the import speed is a snail, it might take 6 hours to import q2 data to the database. Actually, we don't need to index the table before we import the dataset. It only takes about 1 hour to import q2 if the table does not have index. This is only exist in MySQL since Hbase using different import method and Hbase's import is actually a MapReduce job, it can inserting column concurrently.

   3.2 Correctly selecting data types for each column, especially index column. This is also only in MySQL, because Hbase does not have the concept of data type. For example, in q2, we firstly using varchar for userid and time, the comparing of varchar is slower than bigint comparing and date comparing.

   3.3 Increase cache size to allow more data cached in memory. This will will be reduced the lookup time. In MySQL, increasing innodb_buffer_pool_size. And in Hbase, increasing hfile.block.cache.size.

   3.4 Allow more connections at the same time. This will increase the throughput by working on many quries at the same time. In MySQL, increasing the max_connections. In hbase, increasing the hbase.regionserver.handler.count.

   3.5 Indexing the table. This is only need to explicitly set in MySQL. Hbase use B+ tree to do the lookup. So the row-key is the primary index. Without an index, MySQL must begin with the first row and then read through the entire table to find the relevant rows. If the table has an index for the columns in question, MySQL can quickly determine the position to seek to in the middle of the data file without having to look at all the data.

   3.6 Using bloom filter. This is only supported in Hbase. Bloom filters provide a lightweight in-memory structure to reduce the number of disk reads for a given Get operation to only the StoreFiles likely to contain the desired Row.

   3.7 Compression data. Using compression data will reduce the size of database, and will save I/O bandwidth.

   3.8 Tuning JVM GC and heap size. This will reduce region server's JVM garbage collection time. Our setting: export HBASE_REGIONSERVER_OPTS="-Xms8g -Xmx8g -

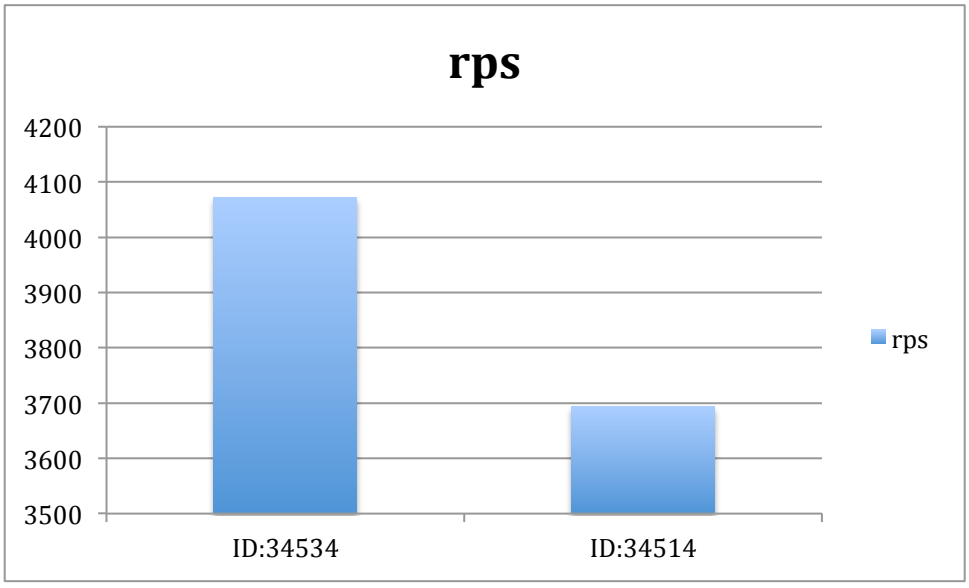
Xmn256m

-XX:+UseParNewGC -XX:+UseConcMarkSweepGC
-XX:+UseCMSCompactAtFullCollection
-XX:CMSFullGCsBeforeCompaction=15
-XX:CMSInitiatingOccupancyFraction=70"

3.9 Region split. In Hbase there might result in “hot region”, region split can split the data to move to a cold region to reduce the load of hot region.

3.10 Balance Load. By doing this, we can avoid load concentrate in the same node. In MySQL, since our front end lives in the same node in backend instance, so we use ELB to do the load balance. In Hbase, we can configure hbase.blanceer.period

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).



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

Our design is tuning for read only tasks, for example, we set hfile.block.cache.size to 0.7. This means that 70% Percentage of maximum heap to allocate to cache for reading. Also, our table design has limit for insert/put request. Because there is only one column for the content to put in.

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

MySQL
Driver - MySQL JDBC driver from official website www.mysql.com
APIs -
a. get connection: Datasource - Connection getConnection() throws SQLException
b. get prepare statement: Connection - PreparedStatement prepareStatement(String sql) throws SQLException
c. set prepare statement parameters: PreparedStatement - void setString(int parameterIndex, String x) throws SQLException
d. execute prepare statement: PreparedStatement - ResultSet executeQuery() throws SQLException
e. get result: ResultSet - boolean next() throws SQLException
String getString(int columnIndex) throws SQLException

HBase:
Driver - HBase 0.94.18 library from apache
APIs -
a. get configuration: HBaseConfiguration - org.apache.hadoop.conf.Configuration create()
b. get connection: HConnectionManager - HConnection createConnection (org.apache.hadoop.conf.Configuration conf) throws ZooKeeperConnectionException
c. get table: HConnection - HTableInterface getTable(byte[] tableName) throws IOException
d. get result: HTableInterface - Result[] get(List<Get> gets) throws IOException

We didn't try other MySQL driver, but we have tried HBase 0.92 driver but we finally decide to use the newer version, since it support some additional operation and has fix some bugs on the previous version.

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.

	ab	c	d	e	f	g	hi
q1	10	80	0	0	0	0	10
q2	10	5	0	75	0	0	10
q3	10	5	0	75	0	0	10
q4	10	5	0	75	0	0	10
q5	10	5	0	60	0	15	10
q6	10	80	0	0	0	0	10

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).

   Facebook:
   NoSQL database fits Facebook Messages. Because for a use case like Messages, clustering the message data on disk by userid as the primary dimension, and time or other properties such as thread id or word in the message (for search index purposes) as the secondary dimension is a reasonable schema choice since we want to able to retrieve the messages for a given user efficiently. The users generate messages at a high volume. So Hbase's high write throughput is better fit for this demand than MySQL.

   RDBMS fits the user information storage. Because the user information is basically a relation of many attributes, and the social network is a relation between users. Thus a RDBMS is better fit for user information storage. For example, a query to present user's birthday, it can be easily accomplished by SQL.

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

   We use the same cluster as our front end system in phase 3, and when developing the back end system, the cluster is also used for testing front end system.

   The total cost (including front end and back end) is $8.

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

    MySQL

- Hidden Features of MySQL
- Book: High Performance MySQL / Blog: High Performance MySQL
- Blog: Xaprb (for MySQL DBAs)

Hbase:

Chapter 14. Apache HBase Performance Tuning:
http://hbase.apache.org/book/performance.html
HBase: The Definitive Guide-Lars George
Blogs:
Hbase performance tuning: http://jm-blog.aliapp.com/?p=975
Hbase: http://flyingdutchman.iteye.com/blog/1858951

Task 3: ETL

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

Read input files from the S3 bucket and get certain columns we want. Then write the records to file, in the format of field terminated by '\t' and line terminated by'\n'. As for Q2, since we are using the Hbase for this phase, the original '\n' and '\r' and '\t' were replaced with "\\n" and "\\r" and "\\t". Besides, the '\' was replaced with "\\". As for Q3, every single line composes of two columns, which are UserID and RetweetUserIDs. As for Q4, every single line composed of 4 columns, which are location, time, hashtag+tweetId, and rank. Finally the files are imported into the database.

In this way, we can speed up the process of importing data to the database, because we have formated the input file. Besides, we also separate the tasks more independently.

b. The type of instances used and justification

In the process of generating the input files, master instance type is c3.xlarge, and core instance type is m1.large, and task instance type is c3.xlarge. In the process of importing data and creating index, the c3.xlarge instance type is used. As for the master and task instance and importing, they are mostly do the computation task, so computation-oriented instance type is used. Besides, the spot price is really low for a long time. As for the core instance, they are mostly to store data, so m1.large instance type is used.

c.The number of instances used and justification

One master instance, 3 core instance, and 10 task instance. Because we have up to a limit of 20 instances. The more task we have, the faster process is. But considering the budget, we are using those instances within the budgets.

d. The spot cost for all instances used

Considering the spot price of m1.large is high, m1.large is on-demand. The prices for c3.xlarge is $0.0321. The time used is less than one hour. The total spot cost without m1.large is $0.3531.

e. The execution time for the entire ETL process

As for the Q1 to Q5, the datasets are reused. As for the query type 5, it takes 3 hours and 9 minutes. It takes 47 minutes to finished it.

f. The overall cost of the ETL process

$8.82

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

As for the query type 5 and 6, the number is 5.

h. Discuss difficulties encountered

Since q1-q4 results are reused, for the query 6, we added an extra column to count the total photo numbers from the smallest userid to the current id. No difficulty is met as to query 5.

i. The size of the resulting database and reasoning
The Raw data is about 30GB. The resulting database is 50.21GB. The database size consists of raw data, redundant backup, and bloom filter.

j. The time required to backup the database

39 minutes

k. The size of the backup

50.21GB

2. Critique your ETL techniques. Based on your experiences over the past 6 weeks, how would you do it differently if you had to do the same project again.

   I would look into the differences between the Mysql and Hbase and decide how to deal with special characters such as '\r', '\n', '\t', '\'. Besides, we would like to research the charset deeply since most of the problems are about the charset.

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

   From my perspective, more instances should be used. However, the running time of the ETL is almost less than 1 hour. It really does not matter.

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

   Yes and I only use the streaming.

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

No, we use build-in tool to load the data.

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

   HBase, because hbase import basically uses MapReduce job to import the data, it is parallelly writing to the database, so it is faster and easier than MySQL. MySQL need to index after importing data finished. It is very slow.

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)?

   Q2:
   advantages: can automatically convert escape characters in the raw data, such as "\\n" will be convert to real line feed "\n" and store in the database.
   disadvantages: none

   Q3:
   advantages: same as Q2
   disadvantages: none

   Q4:
   advantages: can using sql like "rank >= m and rank <= n" to select specific rows
   disadvantages: none

   Q5:
   advantages: none
   disadvantages: none

   Q6:
   advantages: can using sql like "userId >= m limit 1" to select one specific row, even the userId is not exist in the table
   disadvantages: none

   In our design, all queries (Q2 - Q6) are better suited for MySQL. Because every MySQL instance store the whole data, and adding instance could balance the load and getting better performance, while we only have one HBase cluster, and query partial data on the same table may result in hot region, which will still have to handle heavy load. Also, HBase cannot handle some flexible query such as I mentioned above about Q6.

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 MySQL (not HBase)?

Q2:
advantages: none
disadvantages: cannot automatically convert escape characters.

Q3:
advantages: none
disadvantages: same as Q2

Q4:
advantages: none
disadvantages: cannot using condition like “rank >= m and rank <= n” to select specific rows. And need to store ranks in the row key, then expand get request in the front end system, using a bunch of gets to get the result.

Q5:
advantages: none
disadvantages: none

Q6:
advantages: none
disadvantages: cannot using sql like “userId >= m limit 1” to select one specific row. And finally we load the q6 data on the front end (not communicate with back end).

Total disadvantage: cannot balance load due to only one region store parts of a table, every query for that particular part has to access that region.

In our design, all queries (Q2 - Q6) are better suited for MySQL. Because every MySQL instance store the whole data, and adding instance could balance the load and getting better performance, while we only have one HBase cluster, and query partial data on the same table may result in hot region, which will still have to handle heavy load. Also, HBase cannot handle some flexible query such as I mentioned above about Q6.

3. For your backend design, what did you change from Phase 1 and Phase 2 and why? If nothing, why not?

We use HBase as our back end in phase 3, this is because we have been graded on MySQL in phase 2. And since they are very different, we can’t simply conclude the change.

In fact, if we have the choice, we won’t choose HBase as our back end. The reasons are obvious in question 1 and 2.

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?

   For the MySQL design, it won't work well since we use single MySQL back end to store all the data, and thus it has the limitation on scalability. But it's the best way to handle so many concurrent read requests.

   For the HBase design, we can simply add more core nodes to handle big data.

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

   Yes, we use Pylot to generate http load for our web service. Pylot is a free open source tool for testing performance and scalability of web services. It runs HTTP load tests, which are useful for capacity planning, benchmarking, analysis, and system tuning.

   The first reason is that we need Load Generator to warm up ELB. And queuing for the course project test sometimes take a long time, and will result "Did you submit a snail ?" message if ELB is not warmed up.
   The second reason is that we want to warm up our Database. We create a test set from the data parsed by ETL, and randomly generate requests.

More Questions (unscored)

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

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

   Budget management, Hbase(though not very good at tuning it...), MapReduce, Connection pool, Wildfly.

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

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

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

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