HitMan

15619 Project Phase 3 Report

Performance Data and Configurations

Live Test Configuration and	Results
Instance type	**m3.large**
Number of instances	**7**
Cost per hour	**1.005**
Queries Per Second (QPS)	**q1/q2/q3/q4/q5/q6/MIX[q1/q2/q3/q4/q5/q6/]** **score[109/131/106/154/87/98/175/113/150/27/77/83]** **tput** **[15272.2/6618.8/11186.5/7823.8/10081.2/9768.1/5238.5/2063.4/2731.1/568.8/1947.6/2244.7]** **latcy [6/7/4/5/4/5/9/11/8/42/12/10]** **corr [100/99/96/91/72/76/100/99/99/90/73/67]** **error** **[0.00/0.37/1.14/2.52/10.08/1.06/0.00/0.15/0.22/3.96/2.91/1.00]**
Relative Rank [1 - 91] :	Phase 1: **43 rd** Phase 2: **26 th** Phase 3: **12 th**
Phase Score [out of 100]	====== graded ====== Phase 1: **26** Phase 2 Live (selected): **32 (HBase)** Phase 3 Live: **92 (MySQL)** ====== others ====== Phase 2 (Pre-Live): **20** Phase 2 Live (dropped): **2 (MySQL)** Phase 3 (Pre-Live): **82**

Team : HitMan

Members : Zuoyou Gu(zuoyoug) , Jiaqi Luo(jiaqiluo) , Cambi Tao Guo (taog)

HitMan

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

Overall Performance

We’ve made all our live test result into graph in Figure 1. Our cloud service outperformed the requirement in Q1~Q4 as well as Mix-Q1~Q3. However, we encountered some correctness and HTTP error problems for Q5 and Q6. The performance is slightly worse than other questions. Especially Q5, we found one of our instance always gave 500 error. We had this problem before. But when we were about to restart the instance, the server went back in service again… We also had low correctness rate problem. We checked the data in the database, they were correct. But after the front-end got the data, the data might be corrupted. We could not solve this problem. We think this may has something to do with our frontend configuration. Our throughputs for all queries except Q4-Mix were all above the minimum requirements. But those low correctness rate queries greatly undermined the score. For example, Q5. The throughput was pretty high (134%). But the correctness rate was 72%. So the overall score was 87%. Before Q5, we were in the second place, right after FDU, the legend. We were so disappointed. Something out of our expectation was Q4-Mix. Q4 was one of our best seed player. it has the highest score among all queries to MySQL (174% throughput, 154% score). The correctness for Q4-Mix was similar to Q4, but the throughput was much lower. So we think it’s the cache that made Q4-Mix not do well.

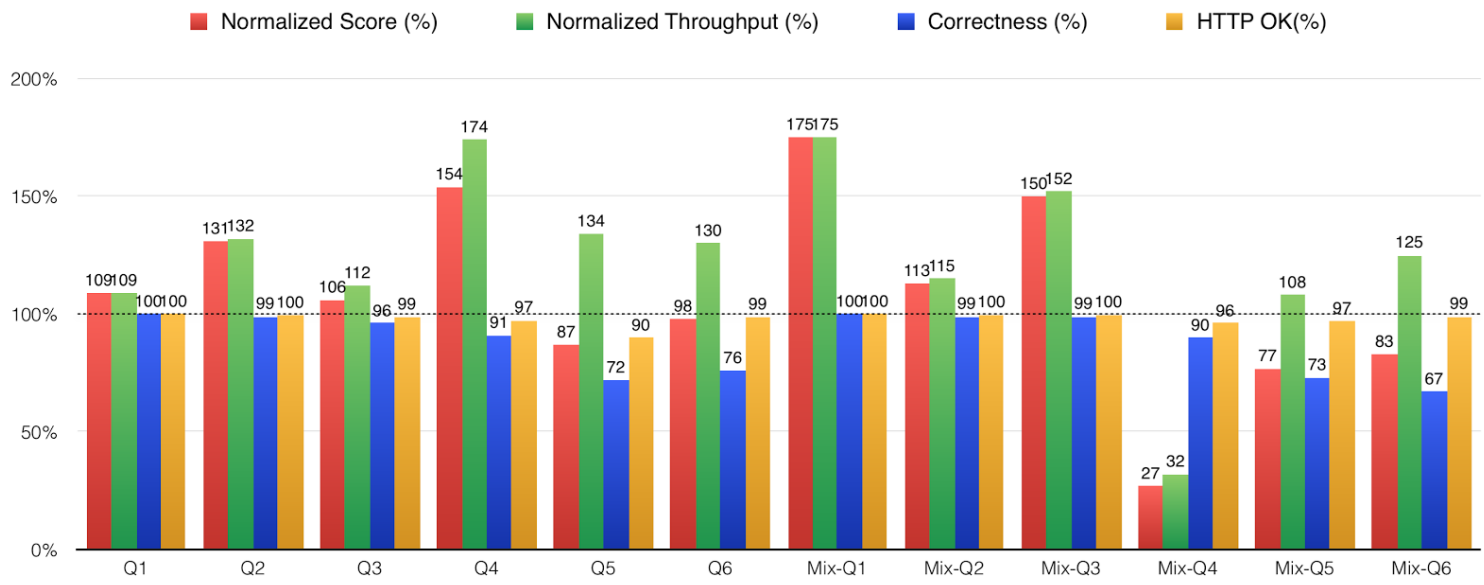


Figure 1. The performance of our web service in each criteria in phase 3 live test

HitMan

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]**

Java Servlet with Tomcat. It is easy to deploy and convenient to use with a stable performance.

Table 1. Tomcat advantages

flexibility	Capable to run Apache on one physical server but run Tomcat service, the actual tomcat JSP and servlets on another machine.
Stability	If a significant failure within Tomcat caused it to fail completely, it would not render the entire Apache service unusable.
Compatibility	Very good HBase connectivity. We have similar issues with other frameworks like Play Framework.
Simpleness	Very easy to build and deploy using ant, which is better than some other frameworks.

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

Table 2. Live test instance information

Instance type	m3.large
Instance number	7

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

We used MySQL connecting pool, setting the max active to 500. Our performance improved a lot.

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

During phase 1, we met a lot of problems and made a lot of bad decisions. We tried a lot to figure out the solution. This is why our Phase 1 cost more than the limit. We had some

problems handling unicode for Q2. We spent a lot of time solving this problem. Even though, the correctness for Q2 MySQL was not 100%. For Q2, the schema was (userid, tweet_time, tweet_id, score, text). We built an index on (userid, tweet_time). The first time we built the index, we didn't know that only attributes with constant length could be index. So we ended up with changing the schema and loading the data again. When converting the csv to HFiles, we found the new line seemed not work properly, many tweets are concatenated together. In Phase 2, we changed Q2 schema to (userid+tweettime, [tweetid:score:text]). From Phase 1, we wanted to use the data right after we get the data from the database. But it was a little tricky for HBase to process new line. But for MySQL, this is not a problem. So we ended up generating different ETL output for MySQL and HBase both. From Phase 2, we found that it is a little hard to tune HBase to achieve a good performance. So in Phase 3, we decided to focus on MySQL only. And also, HBase with EMR is too costy. In Phase 1, we used Python to run the ETL job, but from Phase 2, we switched to Java. It seems Java runs fast. And we once used two m1.large instances to run a ETL job, which ran for 24 hours. But we didn't expect it to run 24 hours. We just sat there to wait it to finish so that we could terminate the EMR right after the EMR job finished. That was a terrible experience!!! So in the following ETL jobs, we used 18 instances. In phase 1. At that time, we thought there were still something wrong with our unicode processing since the correctness was rarely 100%. Until Phase 2 did we find out unicode was not the culprit for the correctness problem. Since we found that the correctness for all queries using MySQL was rarely 100%. But for HBase, the correctness was almost always 100%. So in Phase 3, we tuned a lot of parameters, wanting solve this problem. But this problem was never solved...

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. It does improve the performance. But having 4 instances doesn't mean your performance will increase to four times higher. Our usual score for one m3.large instance is 40-50 for Q3, but with ELB, the score only increased to 106 with 7 instances. Since ELB also takes some time to pass the request and response and also needs time to decide which instance to send request to. ELB will monitor the response, which will also takes some time.

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 did try to explore nginx. But its configuration is too complicated for us to have enough time to make it work..

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

Yes. once the instance is up, the instance is ready to use. We also wrote a script to

auto rebuild and redeploy after modifying its code or configurations.

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 Amazon cloud watch to monitor our front end instances as well the ELB instances to get more information about how good the works. Here is one example of the cloud watch diagram. The cloud watch gives us a good sense of the how our system is performing. The following graph is the CPU utilization graph. All the other instance's CPU is not really fully utilized. One of the instance's CPU utilization was 100% all the time. We used a different configuration and data format for this instance. It was pretty interesting and scaring (since it was live test, we were afraid there would be a lot of errors).

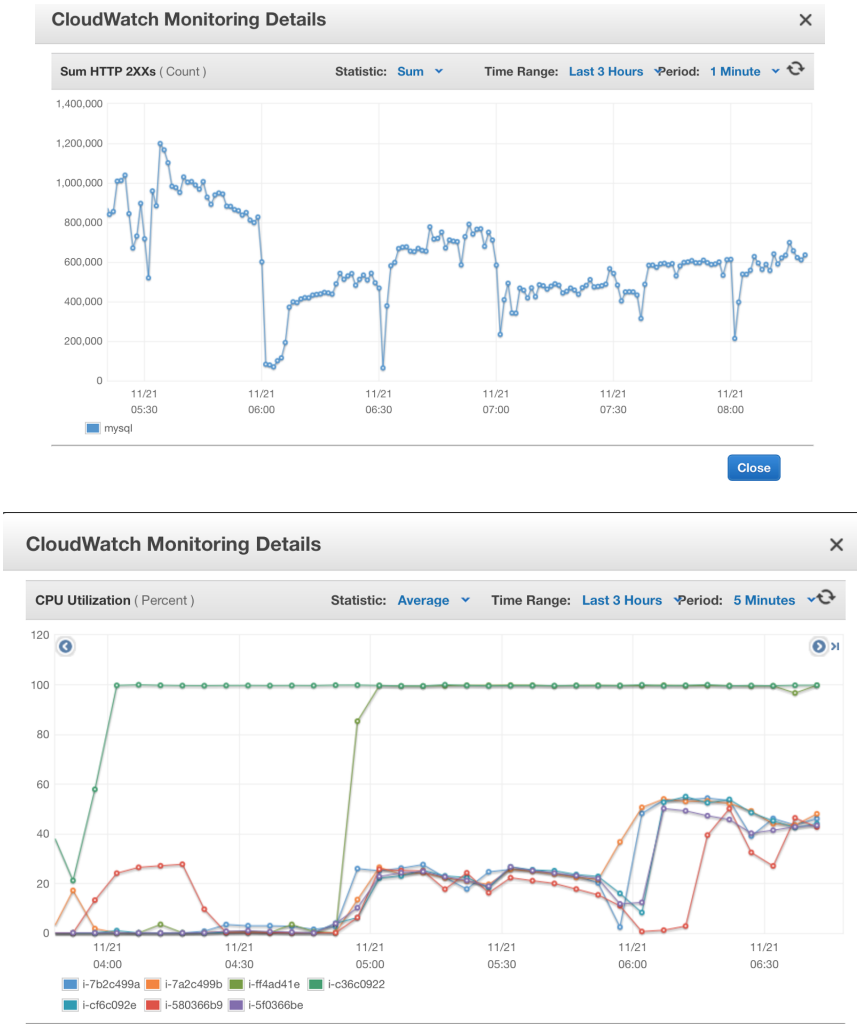


Figure 2. The monitoring of Amazon CloudWatch showing the number of HTTP requests of the load balancer and CPU utilization of each instance

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

In this phase, we didn't spend time on developing front end. So the cost should be 0. We only modified some of the front-end code when testing our Databse.

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

We used http://tomcat.apache.org/tomcat-7.0-doc/introduction.html as the reference to setup the front-end.

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

We kind of do. Since we highly doubt our MySQL correctness problem is related to the front end. For Q1, we first try PHP. But that was really slow for Q1. So we switched to Tomcat with servlet. But tomcat doesn't meet the minimum requirements. So we used hash table to cache the history. If we found out some better front end, we probably wouldn't have to cache history.

HitMan

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.**

   Q2:

```
      date + userID (VARCHAR(40))       |      tweetid + score+tweets content(TEXT)
-----------------------------------------------------------------------------------
      2014-05-07 14:52:48112189480      |      12313:1:hello the world
                                        |      hi the world
                                        |      .....
-----------------------------------------------------------------------------------
      ......                            |      .....
-----------------------------------------------------------------------------------
```

The schemas for MySQL and HBase are the similar. We had two iterations.

At first, we separated the date, user_id, tweet_id, score and tweet content into separate columns. For each query, the database need to compare twice to get the row and the front-end need to do some string concatenation.

The schema of the second iteration is above. After using the new schema as well as improved ELB configuration, the throughput raised from 400 to 4700.

Q3:

```
      userid (BIGINT) PRIMARY KEY       |      retweetids (TEXT)
-----------------------------------------------------------------------------------
      2421587732                        |      342213714
                                        |      (1532300155)
                                        |      .....
-----------------------------------------------------------------------------------
      ......                            |      .....
-----------------------------------------------------------------------------------
```

The schema for HBase is similar. We had one iteration. The schema works pretty well with MySQL. With one m3.large instance, our score was usually 40-50. And the live test also proved this schema worked pretty well (106).

Q4:

date + location (VARCHAR(40))	rank(INT)	tag and ID (TEXT)
2014-05-19 West+Central+Africa	1	RT : 112234
.....	.....	.....

The schema is the similar as HBase. We had two iterations.

The first iteration, we designed have two attributes data+location+rank and hashtags.

The second iteration is the one shown above. The second worked much better. For HBase, the score jumped from 7 to 19. This schema also worked pretty well for MySQL. Q4 alone got 154.

Q5:

userid (BIGINT) PRIMARY KEY	score1(INT)	score2(INT)	score3(INT)
2421587732	10	9	20
.....	.....	.....	.....

We had one iteration. The schema works pretty well with MySQL.

Q6:

userid (BIGINT) PRIMARY KEY	sumpics (INT)
2421587732	342213714
242158773	342213719
.....	.....

We had two iterations. The one above is the first one. The column of sumpics defines as “the number of pictures that are posted by users whose user_id is less than the current user_id”.

For each request, we just need to return the sumpics(userid = n*) - sumpics(userid = m**). Where n* is the smallest userid that is greater than n in the database. And m** is the smallest userid that is greater than or equal to m.

We have a second iteration. We added an auto increasing column id to the database as the primary key and created index on the userid column. However, this no significant performance

improvement so we kept the original iteration.

We also tried memory database engine. However, after applying this engine, the front-end crashes frequently so we failed to use that.

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.**

The most surprising thing we found is the caching or warm-up of the database. This is because when we query the database for first time, the database is on the disk. But the query will make database to load the data into memory. MySQL also does caching, which may also benefit the query. This is the result we got for Q6 with one m3.large instance. We submit Q6 in four consecutive times. The score increased from 0 to 31 at the end. But later the increase was limited and sometimes could be lower. To make the performance better, we used a large instances to have larger memory and warmed up those instances before we submit them. The same thing happened to other queries, too.

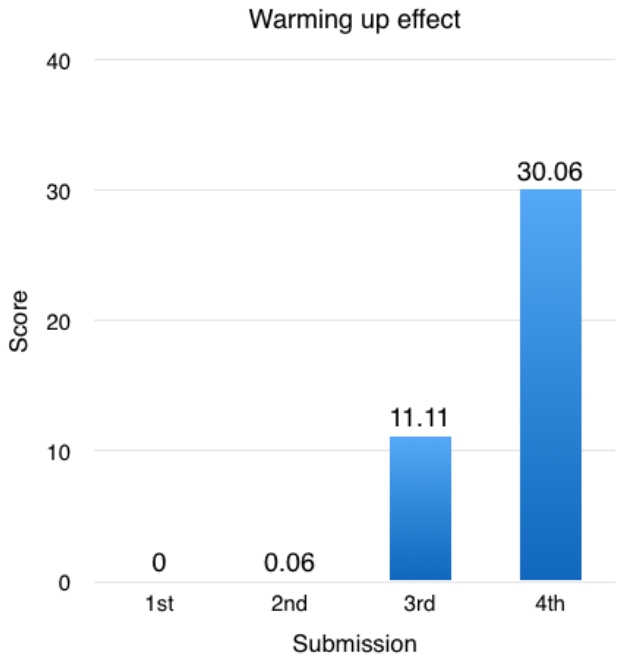


Figure 3. The improvement of performance during the 4 consecutive submissions

Create a new connection to the database is also costly. We solve it by giving a connection pool. After each query, the connection will not be terminated but returned to the pool, which can be used by next query. We used connection pool directly, so we don't have data without connection pool to compare. We also built more backends and adjusted the MySQL

parameters in my.conf.

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).**

(1) Set up a connection pool. In MySQL it configures pool property in datasource. In HBase, alternatively it configures a HTablePool. Before querying, a bunch of connections will be built ,initialized and put into pool, each query will get connection from the pool. And return the alive connection to pool after finishing.

(2) Tune the connection pool parameters, like max active. We can also set similar thing in HBase.

(3) Use better faster disk and larger memory size. This will work for both.

(4) Schema design for both MySQL and HBase. Decrease the number of database queries per each query. Because both querying and string manipulation takes time. It is better to get the response itself from the DB using one database query, or at least fewer and simpler queries.

(5) For MySQL, revise parameters in my.conf. It can alter the table cache, key_buffer, innodb_buffer_pool_size and other parameters. We can also tune similar cache for HBase.

(6) Create index for columns in MySQL database. Index works as a binary tree so the database can quickly find the row of the corresponding query. This only works for MySQL because HBase is already constructed using key-value pairs.

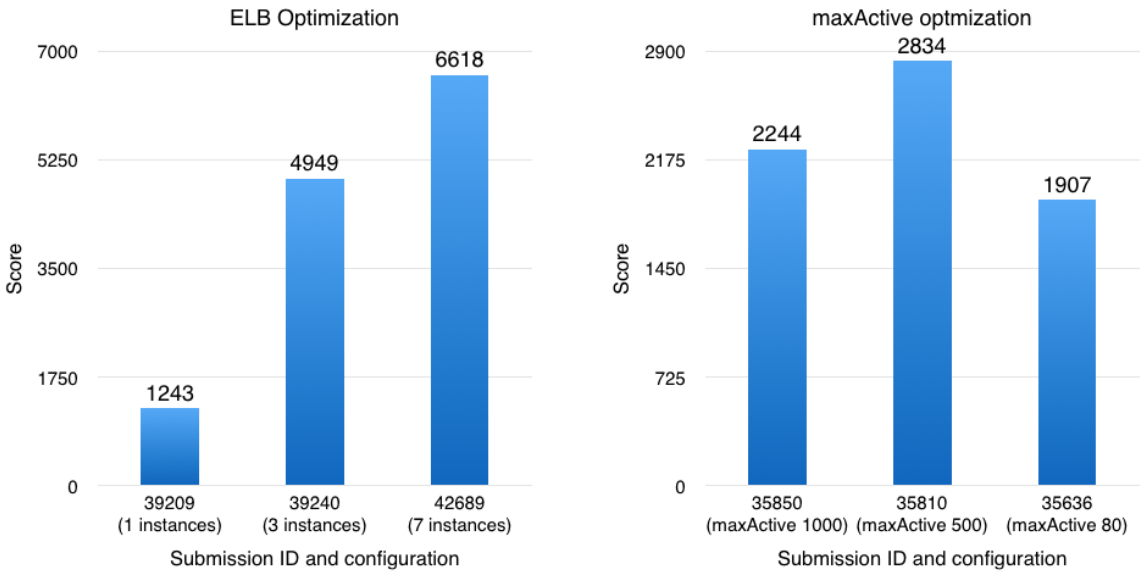
(7) With multiple front-end back-end connected to a load balancer. This can distribute the requests to different server so we can process more queries at the same time

(8) For MySQL, pick the right column type. Use INT other than TEXT or VARCHAR can save space. Use VARCHAR to create columns that need to be indexed because TEXT columns are stored in blob and cannot be indexed. Works only for MySQL

(9) Setup the MySQL database in the same node as the frontend to reduce the time used for network connection. For HBase, considering setting up frontend in the master node. This might be more complicated and may increase instabilities.

(10) For HBase, we can also make the regions evenly distributed across nodes. But this is not available in MySQL.

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).**



(a) *(b)*

Figure 4. (a) The improvement of performance after add an ELB and link multiple instances together using ***technique(7)*** *in the previous question.* x-axis : submission ID for Q2 (1 instance, elb with 3 instances, elb with 7 instances, respectively). y-Axis: the throughput for each submission. *(b) The improvement of performance after tuning the maxActive parameter of the database connection pool using* ***technique(2)*** *in the previous question. maxActive parameter: The maximum number of active connections that can be allocated from this pool at the same time.* x-axis : submission ID for Q2 (maxActive is set 1000, 500 and 80 respectively). y-Axis: the throughput for each submission.

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

No, since we have multiple MySQL servers in service. Update on one server won't be propagated to the other MySQL servers. So the data will be inconsistent.

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

alternatives that you tried?

java.sql package. It has a good compatibility and we can get easy access to it. We didn't tried other alternatives.

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.

We are EXTREMELY sorry but we just didn't profile the backend :P (humorous...), Since we hadn't thought about this. Now let's guess. DB execution takes the most time.

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: The database used to store user account need to be RDBMS. Since the information about the user accounts are very important and requires extremely high consistency and correctness. RDBMS can ensure atomic operations and use rollback to ensure these requirements to be fulfilled.

The tweets should be stored using NoSQL because the volume of data is extremely large. About 500 million tweets a day and about 200 billion tweets a year. If all of them are stored using json format it will be 600TB data a year. The amount of data is not capable for RDBMS to handle effectively. These data doesn't need to be consistent at every moment as long as they will be consistent eventually after some time. So NoSQL will be a perfect solution for this application.

HitMan

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

we use m3.large to develop our back end. We always use spot instance to develop. The bid price is between 0.17 - 0.05. Totally, we spent about 13 dollars for developing. And for live test, we use 7 m3.large, which is 7 * 0.14 = $0.98 per hour.

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

MySQL:

(1) The most useful resource for connection pool is http://tomcat.apache.org/tomcat-7.0-doc/jdbc-pool.html.
(2) resource for handling unicode problem in MySQL: https://mathiasbynens.be/notes/MySQL-utf8mb4

HBase:

(1). The best resource we found is http://databuzzprd.blogspot.com/2013/11/bulk-load-data-in-HBase-table.html#.VG_Y9IvZ6S1. It shows how to load data into HBase.

HitMan

Task 3: ETL

1. **For each query, write about:**

The programming model used for the ETL job and justification

We originally tried EMR using python script to do the ETL job. Then we switched to JAVA EMR to get better performance and compatibility.

Table 3. ETL design

Q2	Mapper	For each tweet, parse the date, userid, tweetid and tweet text. Replace the censored word and calculate the sentiment score. Print all of them
	Reducer	For each key datenuserid, sort the tweets with the same key tweetid. Use predefined field terminator and line terminator to link these information accrding to database schema
Q3	Mapper	For each user A, find the retweetee B. Then print out the A\trB and B\twA
	Reducer	For each key userid, create a retweetee set and retweeter set. Then print out all the retweeter, if the retweeter id is in the retweetee set, parenthesis will be add to the retweeter id.
Q4	Mapper	For each user, get the location and date. For each unique hashtag, print out the location+data, tweetid, hashtag and the smallest start index for this hashtag
	Reducer	For each key location+data, create lists of hashtag and unique tweetids. Then sort the hashtag and unique tweetids and print out them in order.
Q5	Mapper	For each user A, print out A\t1tweetid. And find the retweetee B. Then print out B\t2tweetid;A (We need tweetid, since tweets could be duplicated)
	Reducer	For each key userid, get tweets_number, total_retweetee_number * 3 and total_unique_retweetee_number*10
Q6	Mapper	For each tweet, parse and caculate the number of pictures in it. Print userid as key and number of pictures as value
	Reducer	For each key userid, calculate the sum of number of pictures it posted. Print userid and numpics
	Organizer	Load all the data into memory and sort according to userid. calculate accumulated sumpics defined in the database schema. Print userid and sumpics.

HitMan

The type of instances used and justification

m1.large for all jobs. This is because, Hmm, we never thought about using other instance types.

The number of instances used and justification

Mainly 1 master plus 14 to 18 slaves. A friendly reminder: don't use 1 master plus 2 slaves. It takes 24 hours...

The spot cost for all instances used

The execution time for the entire ETL process

The overall cost of the ETL process

The number of incomplete ETL runs before your final run

Discuss difficulties encountered

(1). handle the unicode problem. Even in ETL, the input and output should handle this problem.

(2). There might be line terminator in the test. We tried to escape them by replacing them with other characters.

(3). During EMR, a '\t' character will be added automatically before the line terminator if there is no one there. Special effort is used to eliminate this effect.

The size of the resulting database and reasoning

For find the size, we used:

```
SELECT table_name AS "Tables",
round(((data_length + index_length) / 1024 / 1024), 2) "Size in MB"
FROM information_schema.TABLES
WHERE table_schema = "test"
ORDER BY (data_length + index_length) DESC;
```

The time required to backup the database

The size of the backup

Table 4. Other ETL information

	q2	q3	q4	q5	q6
number of instances	1+17	1+18	1+18	1+18	1+18
spot cost	0.017	0.017	0.017	0.017	0.015
execution time	1hr20min	1hr40min	1hr15min	1hr31min	1h 30min
overall cost	0.612+	0.017*19*2+	0.017*19*2+	0.017*19*2+	0.57 +

	0.044*18* = 1.404	0.044*19 = 1.482	0.044*19 = 1.482	0.044*19 = 1.482	0.044*19 = 1.406
incomplete ETL runs	1	2	3	0	0
database size	41GB	2.2G	2.4G	3.9G	569 M
db backup time	49min45s	5min34s	2min46s	1min55s	12s
size of db backup	29G	1.2G	1.3G	1000M	353M

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.**

(1) use the hadoop java framework to do the ETL with non-streaming Mapreduce to improve performance.

(2) decrease the numbers of slave nodes in ETL to save money. Even 18 instances can not finish ETL in an hour in phase 2.

(3) Always test a small dataset before running on the entire dataset.

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

(1) Use hadoop mapreduce to do ETL and add more instances to EMR.

(2) Use larger type of instances.

(3) Before running, carefully consider the design of ETL. Use a combiner along with mapper and reducer to speed up the shuffle and reduce the amount of data transmitted via network in the cluster.

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

Yes. We mainly used EMR streaming. Non-streaming would be faster because streaming make it difficult for the entire cluster to get a sense of the overall dataset. And converting types to text stream and parsing the stream take time, increasing the overhead of the mapreduce task.

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

No, we didn't. For MySQL, we simply used the load data command. For HBase, we converted the csv file into HFile first with our own code and then load the HFiles into HBase.

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

MySQL is easier.

We need to use emr to load HBase in hadoop. This process is more complicated. Also, in HBase, we should convert the ETL output file to HFile in file system with a configured and runnable environment. While for MySQL, all we need is just creating the table then use a command to load the file into it.

HitMan

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 consistent, stable and robust. Easier to setup and configure. It supports sort and query cache However, it is very difficult to modify the schema or change the data once established. And MySQL is slow since a jdbc connection is complicated to establish. MySQL doesn't need so many instance as HBase, which will save money. MySQL can handle complicated query. MySQL can handle transaction to ensure data consistency.

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)?**

In our case, we were not able to properly optimize Hbase, so the performance of Hbase for all queries is worse than optimized MySQL. But usually Hbase has a better result when the first query comes than MySQL do. Hbase is more suitable for large amount of data and those data which don't have a structure. And also it's more scalable. In our case, MySQL has a better performance for all queries.

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

We put the backend and the frontend together. Since we won't be doing any write operations during the test, we can use data redundancy to get better performance. And putting backend and frontend together in one instance reduces a lot of network cost.

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?**

The design work fine if it doubles for both MySQL and HBase architecture. However, if it was 10 times larger, the MySQL might be very very slow or even unable to work especially for Q2. The HBase architecture will be more suitable to handle the scale. I think the HBase will work fine given enough budget for data that is 10 times larger.

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

We are EXTREMELY hot, we mean, uncool, so we just didn't do it. And to be honest, we were struggling to meet the minimum requirements even last morning. But thank god, we were finally rescued.

HitMan

More Questions (unscored)

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

   Definitely will try more framework and be greedy (and sneaky) all the time.

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

   (1). Yep, this is the very time we built the whole system from front-end to back-end. I am sorry, tomcat, but we got to be honest, if we have another chance, you are definitely no more our choice.

   (2). Hi mon, I loaded data into HBase for the first time!!!

   (3). My dear remote teammates/teammate!!! When can we meet in person??? Let's talk instead of typing.

   (4). Database optimization!

   (5). The repeating design of ETL. "Hi guys, the ETL for Q4 is done! Oh sh*t, I forgot to remove duplicates.", "Hi guys, the ETL for Q4 is done once again! Oh no, there is a better way to design it..."ETL design. Use some ingenious way to optimize our database for a better performance.

   (6). Live test. Literally, everyone is excited about the live test. We were happy and excited when the score is good. We were anxious when we found there were many http 500. And it's a huge relief when we found out our performance is not satisfying.

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

   Unicode handling. In the beginning, there are bunch of errors in Q2 testing before handling it. We fix it from ETL to backend and frontend. We make sure the read and write in ETL are in UTF-8. In MySQL we set it to UTF8MB4 when creating table and loading data. Also we set the type encoding in response in front end.

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

   Find the specific minute that has the most tweets every day.

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

    In q2, we use the censor words to replace the the line terminators in the tweets in ETL. So we got a lot of f**k and a*s in our csv file. :P

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

    A project that blew me away out of the window… Ahhh… I'm just kidding. We will think about it when we need to find a job next time :P.

HitMan

We appreciate all 619’s staff effort and anyone who ever helped us.