S16 15619 Project Phase 3 Report

Team Name:
elder
Members (First Name, Last Name, Andrew ID):
Jialing Liu (jialingl), Lei Jin (ljin1), Hao Wang (haow2)

Performance Data and Configurations

- Number and type of instances
  - Q1: 6 m4.large
  - Q2: 6 m4.large
  - Q3: 6 m4.large
  - Q4: 6 m4.large
- Cost per hour
  - $0.72
- Queries Per Second (score/tput/ltcy/corr/err)
  - Q1: 5/32936.8/2/100/0
  - Q2: 15/21748.6/2/100/0
  - Q3: 15/13294.7/3/100/0
  - Q4: 15/19731.3/2/100/0
  - MixQ1: 5/11429.7/7/100/0
  - MixQ2: 5/4983/9/100/0
  - MixQ3: 5/4525.3/10/100/0
  - MixQ4: 5/3951.2/12/100/0
- Rank on the scoreboard:
  - Q1: 31
  - Q2: 11
  - Q3: 6
  - Q4: 1
  - MixQ1: 34
  - MixQ2: 8
  - MixQ3: 5
  - MixQ4: 2
  - Overall: 1

Rubric:

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

[Please provide an insightful, data-driven, colorful, chart/table-filled, and interesting final report. This is worth a quarter of the grade for Phase 1. Use the report as a record of your progress, and then condense it before sharing it with us. Questions ending with "Why?" need evidence (not just logic)]

（20%）What improvements have you made to previous queries? (Q1-Q3)
(Name 5 optimizations for each)
Frontend:

1) Choose undertow as our frontend framework. We choose to use undertow since it's faster, lighter and could easily be compiled in gradle and maven project. Also, undertow is better in asynchronous communication.
2) We changed the initial java heap space to 2G and maximum allowed java heap size to 6G for cache and ram for java programmes.
3) We use an ELB in the front end for MySQL system. ELB is great to balance load, which performs much better than the load-balancer implemented by ourselves. Load balancer can better make use of servers. Besides, it can also check the health of servers so as to launch a new server when necessary. If use Round-robin, it means simply allocate the requests to a server, and it can also consider other features such as CPU utilization etc.
4) Use asynchronous communication in undertow by dispatching HttpServerExchange to avoid requests blocking on the front end.
5) Use Gradle to build the whole project. Gradle has advanced task ordering: beyond having full control about the dependencies that are created between tasks, Gradle has powerful language constructs to describe execution order between tasks even if tasks depends not on each others output. This can be modelled with shouldRunAfter and mustRunAfter relationships.

Backend:

1) For concurrency, we used connection pool to do this part. Connection pool contain multiple pre-connection which were connected before the client send requests.
2) Search for primary key. Primary key is unique in each table, mysql automatically build index on primary key in order to improve the efficiency of searching.
3) Change query_cache_size. Query_cache_size cache the query and it's answer into memory which can be retrieve fast when request again.
4) Change query_cache_type. Query_cache_type is 0 as default. So we change it to 1 to enable query cache.
5) Use index. In Q3, we set union index on userid and date so that union range query could be faster.

（30%）Q4: Front end

Questions

1. (10%) Explain your configurations of your front end system. (instance type, number of instances, configurations)

Instance type: m4.large

Number of instances: 6

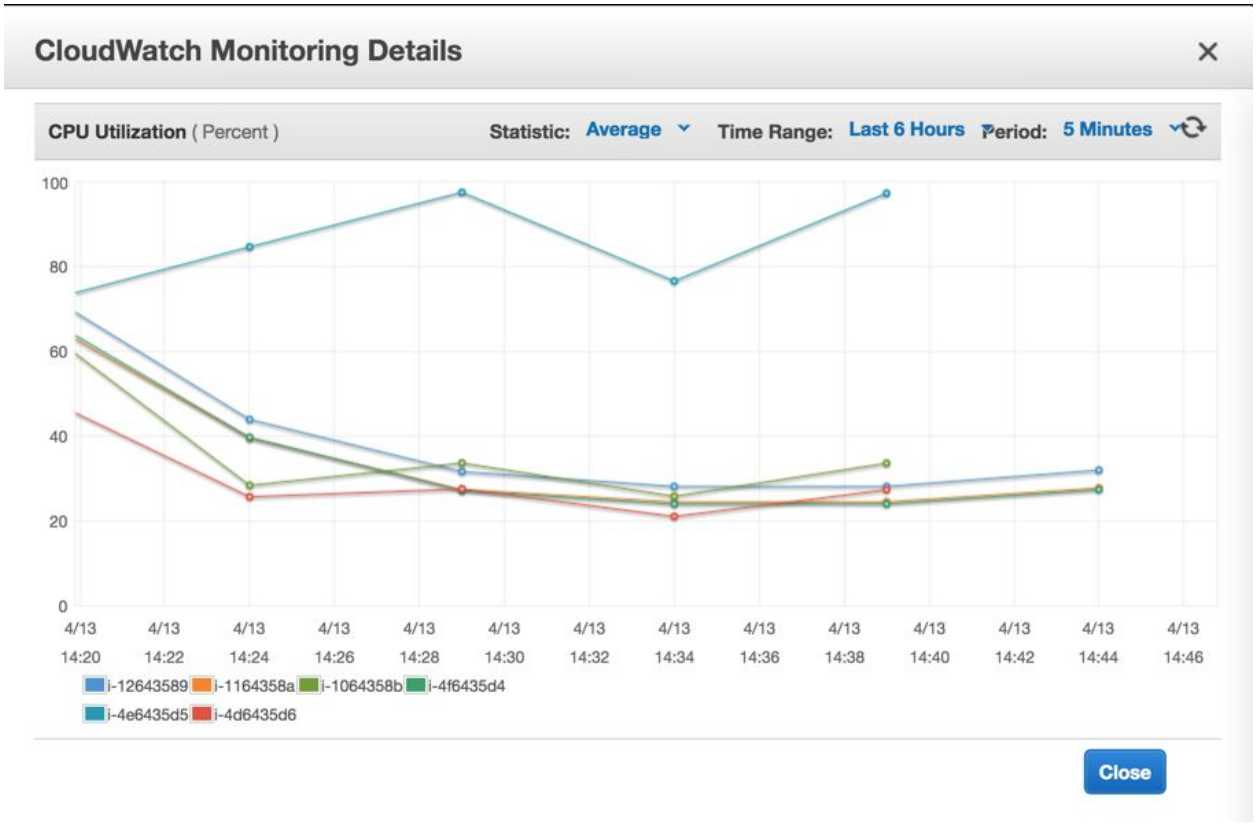
Configurations:
In gradle.properties, we set
org.gradle.daemon=true
org.gradle.jvmargs=-Xms2048m -Xmx6144m -XX:MaxPermSize=1024m
-XX:+HeapDumpOnOutOfMemoryError -Dfile.encoding=UTF-8

2. (15%) What have you done on your front end to improve write performance?

Since we do sharding in the back end part, we set a flag with each front end server, i.e. flag from 0 to 5. Then, we mod received tweetid by 6 to get a newflag. If the newflag equals flag, then the server itself pass the data to its database to do write operation. Else if newflag is not equal to flag, the server will forward this request to the relative server. Meanwhile, the original server will get response from the other server then respond to the ELB. Also we set the read timeout threshold to avoid waiting response for a long time when forward our request to another server. To prevent port blocking, we use some ports other than port 80 to listen forward request.

3. (5%) What are some other improvements you can think of but haven't tried?

Other improvements include writing the load balancer by ourselves. We've tried and could get full score just a little more than the target. But the CPU utilization for the load balancer is much higher than other servers. Specifically, the CPU utilization for the load balancer and other servers are about 85%~100% and 20%~35% respectively, which are shown in the following picture.



***Note: blue line - load balancer; other color - other servers**

CPU utilization evenly distribution taken into account, we utilized aws ELB finally.

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

http://undertow.io/undertow-docs/undertow-docs-1.3.0/index.html
https://docs.gradle.org/current/javadoc/
http://www.cloudera.com/documentation/enterprise/5-5-x/topics/cm_intro_api.html
http://stackoverflow.com/
http://stackoverflow.com/questions/14763079/what-are-the-xms-and-xmx-parameters-when-starting-jvms
https://www.digitalocean.com/community/tutorials/how-to-use-apachebench-to-do-load-testing-on-an-ubuntu-13-10-vps
https://en.wikipedia.org/wiki/Base64

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

（40%）Q4: Back end (database)

Questions

1. (5%) Describe your schema. Explain your schema design decisions.
    a. Would your design be different if you were not using this database?
    b. How many iterations did your schema design require?
    c. 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.)

MySQL schema:

tweetid: bigint(20) unsigned, NOT NULL, PRIMARY KEY userid: TEXT username: TEXT thetimestamp: TEXT thetext: TEXT hashtag: TEXT ip: TEXT coordinates: TEXT repliedby: TEXT reply_count: TEXT mentioned: TEXT mentioned_count: TEXT favoritedby: TEXT favorite_count: TEXT useragent: TEXT filter_level: TEXT lang: TEXT
ENGINE = MyISAM

Reason:

1. PRIMARY KEY for tweetid: we need to insert or update or get the field of a specific tweetid. So we set it as the PRIMARY KEY for faster write and read operations.
2. TEXT data type for all fields other than bigint(20) unsigned for tweetid: tweetid is given to be less or equal to 18 digit long. So we use bigint(20) unsigned to store it. For other fields, however, since we directly stored the encoded form of text into database for unnecessary string processing, the strings for some fields might be too long to be stored as varchar type.
3. Use thetimestamp and thetext to indicate timestamp and text in database: since timestamp and text are reserved words in MySQL, it would bring trouble if we didn't do so.
4. Use MyISAM as the engine for Q4: we have MySQL 5.5 installed on our instances. So the default engine is Innodb. We tried both to compare the performance and Innodb is better for read-only table such as Q2 and Q3. However, MyISAM performs

better on Q4.

Iterations: depends on the number of set requests for distinct tweetid.

2. (9%) What was the most expensive operation / biggest problem with your DB that you had to resolve for Q4?
    a. Why does this problem exist in this DB? How did you resolve it?
    b. Plot a chart showing the improvements with time.

MySQL: Concurrency. We first used single thread to do query, but it is extremely slow. Then we created multi-thread by ourselves, but it is still very slow. Finally we used connection pool and it is much faster than before.

Before using connection pool	After using connection pool
Id 93404 with score 13.77 Id 93409 with score 10.59 Id 93413 with score 12.56	Id 93513 with score 18.6 Id 93520 with score 27.92 Id 93524 with score 30.34

3. (10%) Explain (briefly) **the theory** behind (at least) 3 **write** performance optimization techniques for your databases. **How are they implemented in the database?**

MySQL
1) Concurrency. We used connection pool to do this part. Connection pool contain multiple pre-connection which were connected before the client send requests.
2) Search for primary key. Primary key is unique in each table, mysql automatically build index on primary key in order to improve the efficiency of searching.
3) Change query_cache_size. Query_cache_size cache the query and it's answer into memory which can be retrieve fast when request again.
4) Sharding rather than replication for Q4. Considering that there might come a large number of set requests, insert or update and read operations would be slower and slower as the table grows larger and larger. Besides, MyISAM permits data and index files to grow up to **256TB** by default. So we did sharding for six tables on six different servers.

4. (5%) Plot a graph showing results with/without each individual optimization that you used. Extremely impressive will be a timeline of rps v/s submission id (mentioning which optimization was in use at that time).

	Submission id	Without optimization	With optimization
MySQL	Connection pool	Id 93413 with 12.56	Id 93524 with 30.34

		score	score
	Search primary key	Id 93513 with 18.6 score	Id 93589 with 28.05 score
	Increase query_cache_size	Id 94695 with 22.79 score	Id 94724 with 44.57 score
	Sharding databases	Id 94695 with 22.79 score	Id 95407 with 62.12 score

5. (5%) Did you face any machine failure during live test? What would you do for fault tolerance?

Fortunately, we didn't face any machine failure during live test. Only one server's CPU utilization was a lower than others in Q2. Then we restarted the server, but its CPU utilization didn't improve a lot. So we think the reason might be that EBS warm-up for this machine was not enough.

6. (2%) Which API/driver did you use to connect to the backend? What features/configurations are needed to achieve good performance?

We use JDBC to connect to MySQL server. JDBC is part of the Java Standard Edition platform, from Oracle Corporation. It is a industry standard for database-independent connectivity between the Java programming language.

7. (2%) How did you profile the backend? If not, why not? Given a typical request-response for query 4, 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 9x3 table is one possible representation.

MySQL Q2:

Factors	Latency percentage

Load Generator to Load Balancer	cannot measure
Load Balancer to Web Service	0
Parsing request	1
Web Service to DB	2
At DB (execution)	80
DB to Web Service	2
Parsing DB response	15
Web Service to LB	0
LB to LG	cannot measure
Total	100

MySQL Q3:

Factors	Latency percentage
Load Generator to Load Balancer	cannot measure
Load Balancer to Web Service	0
Parsing request	0
Web Service to DB	1
At DB (execution)	93
DB to Web Service	1
Parsing DB response	5
Web Service to LB	0
LB to LG	cannot measure
Total	100

MySQL Q4:

Factors	Latency percentage
Load Generator to Load Balancer	cannot measure
Load Balancer to Web Service	0
Parsing request	1
Web Service to DB	2
At DB (execution)	90
DB to Web Service	1
Parsing DB response	0
Web Service to LB	0
LB to LG	cannot measure
Total	100

8. (2%) What was the cost to develop your back end system?

We used 6 m4.large instances as our back end system. The cost was 6 * $0.12 = $0.72 per hour.

9. What were the best resources (online or otherwise) that you found for your backend?

http://stackoverflow.com/questions/16377932/mysql-behavior-of-on-duplicate-key-update-for-multiple-unique-fields
http://stackoverflow.com/questions/21239979/how-to-clear-mysql-query-profiles
http://stackoverflow.com/questions/11768127/use-limit-in-a-mysql-insert
https://bugs.mysql.com/bug.php?id=9676
http://stackoverflow.com/questions/16609391/set-max-rows-insert-limit-mysql
http://dba.stackexchange.com/questions/53831/large-insert-into-select-from-gradually-gets-slower
http://stackoverflow.com/questions/5253302/insert-into-select-for-all-mysql-columns
http://stackoverflow.com/questions/5924115/using-limit-1-in-mysql
http://stackoverflow.com/questions/3536103/mysql-how-many-rows-can-i-insert-in-one-single-insert-statement
http://stackoverflow.com/questions/11768127/use-limit-in-a-mysql-insert
http://kvz.io/blog/2009/03/31/improve-mysql-insert-performance/
http://www.mysqlab.net/knowledge/kb/detail/topic/performance/id/5114

http://dba.stackexchange.com/questions/14775/performance-settings-for-myisam-tables-keep-everything-in-memory
https://mariadb.com/kb/en/mariadb/memory-storage-engine/
http://stackoverflow.com/questions/9819271/why-is-mysql-innodb-insert-so-slow
http://stackoverflow.com/questions/2520357/mysql-get-row-number-on-select
http://stackoverflow.com/questions/10457458/select-specific-row-from-mysql-table
http://dev.mysql.com/doc/refman/5.6/en/range-optimization.html
http://stackoverflow.com/questions/5161291/mysql-optimizing-query-for-records-within-date-range
http://stackoverflow.com/questions/8881331/mysql-optimizing-the-search-on-range-of-records
http://dev.mysql.com/doc/refman/5.1/en/index-hints.html
http://www.tomcatexpert.com/blog/2010/04/01/configuring-jdbc-pool-high-concurrency
http://aws-labs.com/ubuntu-install-java-8/
https://www.linode.com/docs/databases/mysql/install-mysql-on-ubuntu-14-04
https://www.digitalocean.com/community/tutorials/how-to-use-mysql-query-profiling
https://www.percona.com/blog/2015/01/02/the-mysql-query-cache-how-it-works-and-workload-impacts-both-good-and-bad/
http://www.mkyong.com/java/find-out-your-java-heap-memory-size/
https://www.linode.com/docs/databases/mysql/tuning-your-mysql-database
http://serverfault.com/questions/78786/tuning-and-understanding-table-cache-in-mysql
http://webrewrite.com/how-to-check-ram-in-ubuntu-11-1012-04-using-terminal/
http://stackoverflow.com/questions/20259249/what-is-query-cache-prunes-per-day-in-mysql
http://www.hardwaresecrets.com/how-to-optimize-a-mysql-server/6/
https://easyengine.io/tutorials/mysql/mysqltuner/
http://stackoverflow.com/questions/3753504/mysqltuner-suggestions-and-changes-to-my-cnf
https://stanford.edu/dept/itss/docs/oracle/10g/appdev.101/b10826/sdo_objrelschema.htm#i1004730

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

(10%) General Questions

1. (2%) Would your design work as well if the quantity of data would double? What if it was 10 times larger? Why or why not?

I think there should not be much different, for we used large-scale database systems to store those data, instead of saving them in plat files or RAM. So this design of front-end and back-end server is enough and it does not make much difference of performance. Though there is a 256TB storage limitation for MyISAM engine databases, we used it only in Q4's table and the space is enough. Therefore, the data doubling is not a big deal for our web system.

2. (2%) Did you attempt to generate load on your own? If yes, how, how is it helpful?

Yes. We used a browser to send HTTP GET request, just like the load generator does, in order to see whether the result is bad or not. And observe the time consumption of handling single query. It was helpful to check whether a single query is correct.
Besides we also developed our own load generator by write bash script and use multiple threads to generate HTTP request to our web application. It was pretty helpful to keep elb warmed until live test.

3. (2%) Describe an alternative design to your system that you wish you had time to try.

We tried to use MySQL memory engine for Q4, since MySQL with such engine could store data in memory so that the data could be inserted, updated or read extremely fast. However, TEXT data type is not supported in this database engine and we need to use varchar. But text field might be too long for varchar(255) without decoding. So we decoded the payload before insert and update operations. While, new problems came out, such as quotation or percentage marks existed in the text. Then we replaced all them with backslash and themselves. But another problem emerged, the table was full with only 18150 rows inserted. Then we change the parameter max_heap_table_size in the configure file of MySQL. However, problem still existed.

We wish we had time to keep trying this kind of MySQL engine. But it seemed to require precise estimation of data size as well as the ability for memory control which we were lack of. Limited time taken into consideration, we gave up finally and turned to MyISAM engine.

4. (2%) Which was/were the toughest roadblock(s) faced in Phase 3?

We consider the toughest roadblocks faced in Phase 3 to be forwarding HTTP requests to other servers.
When using one instance (main server) as the load balancer as well as the front end server for its sharding-part database, we simply used HttpURLConnection to forward those requests which doesn't belong to it to other server's port 80. It works pretty well on Q4 part. But it meant that we need to sacrifice Q1 Q2 Q3 which could get high scores using an

ELB.
So we used an ELB to distribute load for six instances. Though it worked well at first, but occasionally, one or more instances were caused to “die” for some unknown reason. And when we cut down one of the server, other servers would raise NullPointerException with ERROR: Blocking request failed HttpServerExchange{ GET /q4}. It took us quite a while to figure out that this situation was due to port confliction. So we modified our codes and on one front-end instance, we created multiple servers listening to different port to receive requests from other servers without conflict.

5. (2%) Did you do something unique (any cool optimization/trick/hack) that you would like to share with the class?

Start early. We made nearly every possible mistake, such as string split without using an integer index, not using raw URL parameters, opening unnecessary threads in undertow, memory blast, table full… But we had time to fix those problems and learned a lot during the process.
Do not procrastinate. Since we’ve done everything well in phase 2, we didn’t need to worry about Q1 Q2 and Q3 anymore. Therefore, pressure was much less and budget was enough.