Team Name: MatrixHacker

Team Member and Andrew ID: Tao Yu (taoyu) Wendi Zhang (wendiz) Tianwei Li (tianweil)

15-619 Project Report

1 Statement of Assurance

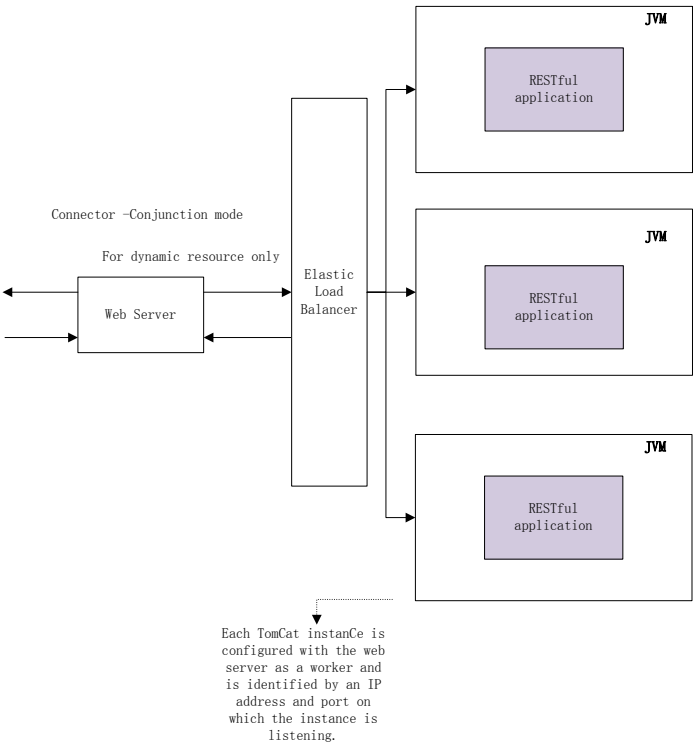
We certify that all of the material that we submit is original work that was done only by our team.

2 Web Service Overall Design

2.1 Summary of web service architecture

Please provide an overview of your web service architecture, the functionality of each component, the web server platform used (Jungle, Tomcat, Apache PHP, etc.).

We chose to use Tomcat, it provided us a pure HTTP web server environment so that we don't need to build the whole HTTP server by ourselves and only need to focus on how to respond each HTTP request. Since we are using JAVA to develop a RESTful web server, we need to use Tomcat behind Apache server to forward the requests.



2.2 Database Selection and Database Schema

We have completed all the development from step1 to step5with HBase and MySQL. After comparing the performance in online test, we choose MySQL database for the live test.

Below is the performance data in system test (online test)between HBase and MySQL:

Throughput(qps)	q1	q2	q3	q4
MySQL	5397.7	3322.5	1809.5	3954.3
HBase	4987.4	3845.6	789.6	2429.2

- ✧ We all make full use of the budget with HBaseand MySQL.

There are 4factorsto consider:

Dataset size: After the ETL section, the dataset size is 5G, for a majority of small-to medium-volume applications, it is better to use RDBMS solutions—MySQL. Meanwhile, the dataset and the table schema are fixed with no write operations.

Speed: If we use HBase as the back end, there exits network delay between the front and back end.

Query Type: MySQL can meet the query type of q1, q2 and q3efficiently. For q3 example, HBase's rowkey is sorted lexicographically, we should query like this "scan 'table', {STARTROW =>minimum userid, ENDROW =>maximumuserid}", it is less efficient than MySQL.

Budget: Based on the budget of back end with 1 dollar, we can only use 3 large instances with one EMR. In contrast, we can combine back and front end together with the budget very low.

Considering 4 factors discussed above and the online test performance, we decide to choose MySQL for live test.

3 Implementation

3.1 Step 1: Front End

- ➢ **The design of the front end system**

We implemented the front-end system by tomcat and we build a restful web service by java program to handle the http requests.

- ➢ **The type/number of instances used and justification**

We use MySQL database, combining the front end and back end. So we implement the whole system with 1 ELB and 9 m1.medium instances.

There are three choices for us:

1. ELB + 19 m1.small instances.
2. ELB + 9 m1.medium instances
3. ELB + 4 m1.large instances.

They are all under the budget. We tested the three schemes by the live test web page and found the second one has the best performance.
We also considered include Auto Scaling Group and Cloud Watch to improve the performance for heavy network load. But it is difficult to find out the scale in and scale out threshold and the delay of scale in and scale out my lead to lower throughput and higher cost.
So at last, we selected the ELB + 9 m1.medium instance scheme.

- **Other configuration parameters used**

We optimized the tomcat server by increase the memory limit used by tomcat and carefully design the front end program to let the entire request share the same connection to the MySQL server. We also implemented the connection pool to maintain a number of established connections to MySQL server.

- **The cost per hour for the front end system**

Because we use MySQL database, we combine the front and the back end. So we implement the whole system with 1 ELB and 9 m1.medium instances. The front end and back end share the same instances. We calculated accurately the total cost is less than 1.2 $ per hour.

Cost: 0.025 + 0.12*9 + 5 * 0.008 = 1.145<1.2

- **The total development cost of the front end system**

We implemented the front-end system in our personal computer first and installed the MySQL too. We use a small table to simulate the behavior of 4 queries and debug the front-end program. After the program works fine, we move it to the instance and created an AMI. So the cost to develop and test the front-end system is very small, under 0.5 $. We only use ELB + 2 m1.micro instances to test the front end system (by testing the q1 query).

- **The difficulties during you implementation and how you solved them**

The first difficult to develop the front-end system is that we are not familiar with the tomcat and web service programming. We spent lots to time to search on Internet and try to setup the web server by install and configure apache and tomcat.
Another difficult is to write the web service program by java. The format of web server program is different with the traditional application program. The official web page help document of myeclipse helps a lot.
The third difficult is to connect to MySQL server from the front program. We use the JDBC interface to communicate between front-end to back end. There are several tricks to implement it.

For example, we establish the connection once and reuse it afterwards. And the statement must be re-established for every query; otherwise there will be error when trying to get the database result in a high throughput. Then the error rate will not be zero.

- **The throughput of your front end system from test service (testid is required)**

The throughput of the front-end system is tested by q1 request.
The throughput is 10931.2 qps, the testid is 609.

3.2 Back End

- **The design of the back end system**

We use MySQL database in backend and use 3 tables to support q2, q3, q4 respectively.Through the function of creating instance AMI image, we could only load data into MySQL once and use the image to launch other instance with data.

- **The table structure of the database and justify your database schema**
  - ✓ 3 tables for each query

```
+------------------+
| Tables_in_twitter |
+------------------+
| tweetsq2          |
| tweetsq3          |
| tweetsq4          |
+------------------+
```

  - ✓ Table tweetsq2 for the query 2 (Text of tweets)

```
mysql> describe tweetsq2;
+-----------+-------------+------+-----+---------+-------+
| Field     | Type        | Null | Key | Default | Extra |
+-----------+-------------+------+-----+---------+-------+
| create_at | varchar(35) | YES  |     | NULL    |       |
| tidtext   | blob        | YES  |     | NULL    |       |
+-----------+-------------+------+-----+---------+-------+
```

  - ✓ Table tweetsq3 for the query 3 (Number of tweets)

```
mysql> describe tweetsq3;
+--------+------------+------+-----+---------+-------+
| Field  | Type       | Null | Key | Default | Extra |
+--------+------------+------+-----+---------+-------+
| userid | bigint(20) | YES  | MUL | NULL    |       |
| count  | int(11)    | YES  |     | NULL    |       |
+--------+------------+------+-----+---------+-------+
```

✓ Table tweetsq4 for the query 4 (Who retweeted a tweets)

```
mysql> describe tweetsq4;
+----------+------------+------+-----+---------+-------+
| Field    | Type       | Null | Key | Default | Extra |
+----------+------------+------+-----+---------+-------+
| o_userid | bigint(20) | YES  | MUL | NULL    |       |
| userid   | text       | YES  |     | NULL    |       |
+----------+------------+------+-----+---------+-------+
```

- **The type/number of instances used and justification**

  We use 9 medium instances with 20Gsizeroot, we expand the root volume to 20G to support the MySQL database tables.

- **The cost per hour for the back end system**

  Because we use MySQL database, which combine the front and the back end together, we implement the whole system with 1 ELB and 9 m1.medium instances. The front end and back end share the same instances. We calculated accurately the total cost is less than 1.2 $ per hour.

  Cost:0.025 + 0.12*9 + 5 * 0.008 = 1.145<1.2

- **The spot cost for all instances used**

  7 m1.medium * 0.015$/h * 10h = 1.05$

- **The difficulties during you implementation and how you solved them**

  Encode: We should keep the original encoding of the text of tweets. As a result, when we load data into MySQL database, we should add "*load data local infile 'tweetsq2.csv' ignore into table tweetsq2 fields terminated by '\t\t'* ***escape by "\"***"to keep the encoding of the text.

  Table reconstruct: To improve the qps of query, we reconstruct the table schema. For q2, we combine the tid+text together and sorted by tid for one value in ETL. For q3, we count the total tweets of each userid in ETL. For q4, we combine the retweet userid together and sort them first before inserting to database.

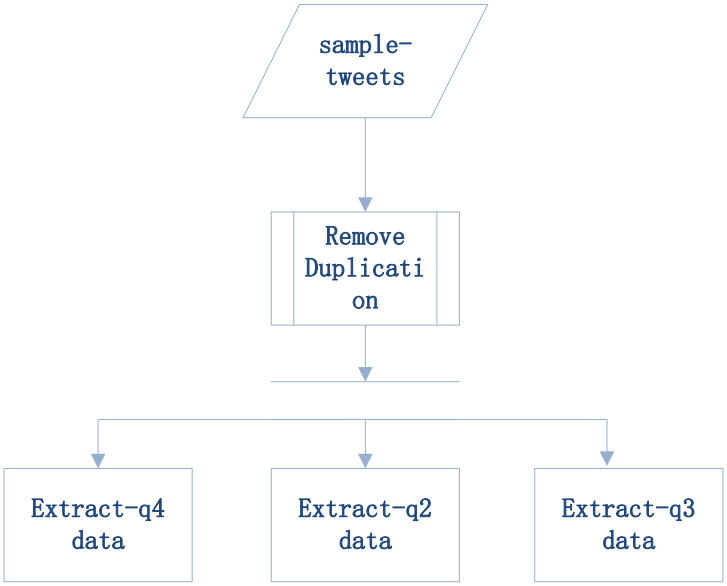
- **The total development cost of the back end system**

  7 m1.medium * 0.015$/h * 20h = 2.10$

3.3 ETL

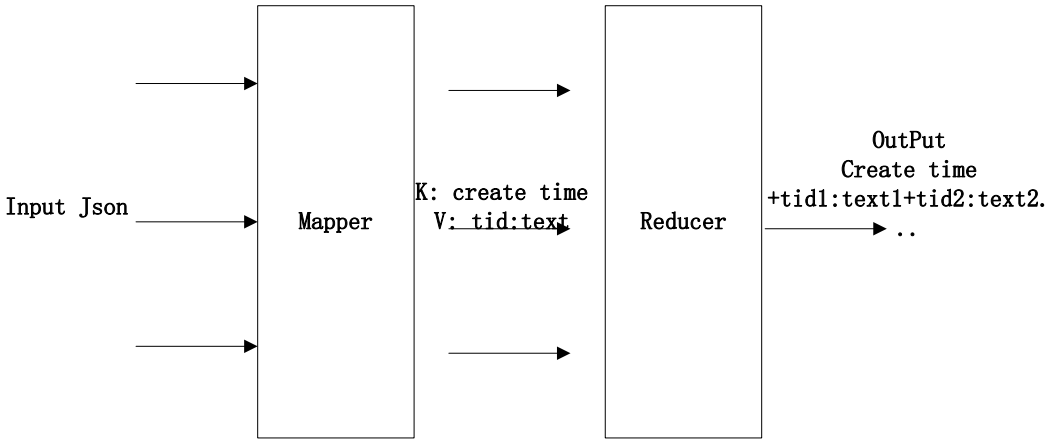
- **The programming model used for the ETL job and justification**

We choose to use Amazon EMR tool to do the ETL job. The job flow is shown as below:

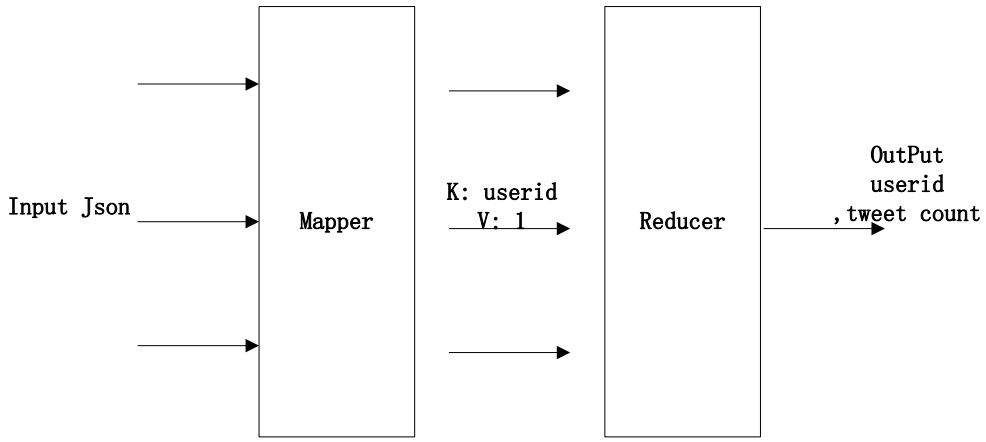


First we remove duplicate tweets in the sample tweets files, and use the result as input of data extraction process and extract data for different queries in the format, which can easily load into database.

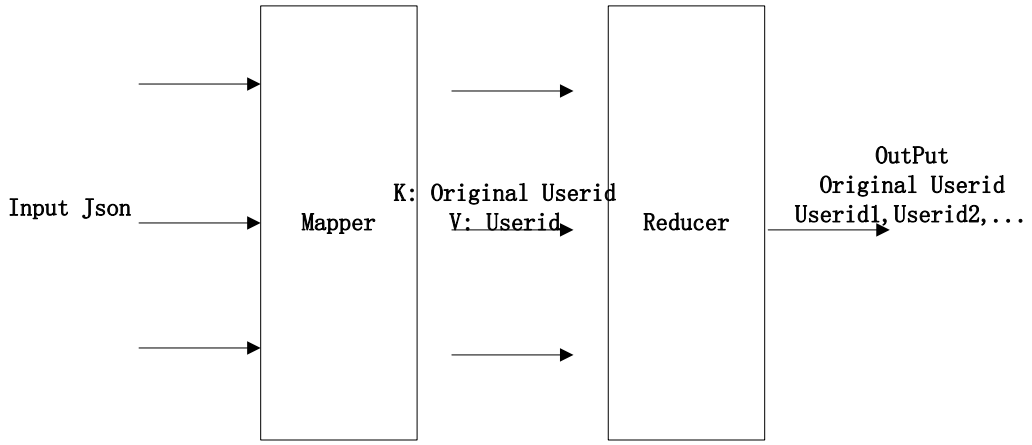
q2 mapreduce model:



q3 mapreduce model:



q4 mapreduce model:



The reason why we choose this kind of program model is, after finishing the ETL process we got the data that has exactly the same format as we expect in MySQL database and HBase database, thus applying this interface will allow us to load data into database very efficiently.

- **The type/number of instances used and justification**

  We choose to launch m1.large instance because at spot price, there's not huge differences between m1.large and m1.medium, since using m1.large can save much time, it's saving our money to some aspects. We choose to launch 6 m1.large instances at a time because we found at this point each map-reduce job can be done within one hour, thus the cheapest approach.

- **The spot cost for all instances used**

  6 m1.large * (0.026+0.06)$/h * 4h = 2.064$

- **The execution time for the entire ETL process**

  Remove duplicate: 90 minutes.

  Query2 data process: 30 minutes.

  Query3 data process: 10 minutes.

  Query4 data process: 10 minutes.

  Data transfer: about 30 minutes.

  So including some interval time between each task the whole ETL process can be done within 4 hours.

  However, for some reasons problems like duplication of tweets , the format problem of query2 and definition of original user show up, we have to redo the ETL process again and again, so it might take about 20 hours for entire ETL process.

- **The overall cost of the ETL process**

  6 m1.large * (0.026+0.06)$/h * 20h = 10.32$

- **The number of incomplete ETL runs before your final run**

  About 50 times, if compile errors count.

- **Discuss difficulties encountered**

  Design: The difficulty is in the design process, when we have to decide which kind of interface I'm going to provide to for further use, the rest part is the simple coding job.

  Problem in q2 data process: There's one thing to notice is that if we use json object in java, it will automatically convert encoding characters in text field to utf-8 characters, which is not we want. So in ETL of q2 we have to extract the text field by matching certain pattern in source file.

  Problem in using Hadoop: The problem is that since we use our reducer code as the combiner, we have to justify the output key, value pair of the reducer to match its input format. Another problem is that we have tried to load data in HBase, if we choose to do this process by using EMR, it might fail due to jobs unable to report status in 600 seconds. A solution is using ImportTsv plus LoadIncrementalHFiles to load our data into HBase instead. It will significantly save our time in this process.

- **The size of the resulting database and reasoning**

  The size of database of MySQL is 5GB. The table for query 2 to is about 4.7GB, query 3 is about 200MB and query 4 is about 100MB.
  The total size of HBase database is about 40GB, maybe it is because all data in HBase is stored in hexadecimal byte format, and it stores huge amount of information such as timestamp and old data.

- **How did you backup your database and how much time did it take to backup/restore**

  We chose to backup our database in csv format on S3. The size of backup file is about 5GB, so assuming the average uploading speed is 20MB/S, it will take 5*1024/20 = 256 seconds.
  Since we have tried to use HBase to support our back-end, we have a backup of HBase data as well. We use CLI command to add a backup step in our job flow, and it will take about 2-5 minutes for the backup or restore process.

- **The size of backup**

  5GB for MySQL. 40GB for HBase.

3.4 Step 4: System Test

- **The design of the communication between front end and back end**

  We chose to use JDBC at front end to connect MySQL database at back end.

- **The difficulties encountered and how you overcome them**

  At first because of the bad design, we have to connect to database every time we handle a new query, which introduce huge amount of latency. At last we decided to initiate the connection to the database only once at the starting point in our server, and use connection pool to reduce connection overhead between database.

- **The AWS resources utilized to realize your system on AWS**

  We used AWS EC2 and ELB service to implement our system.
  For the given workload

  - **The maximum throughput of your overall system for q1, q2, q3 & q4**

    max q1: 5397.7

    max q2: 3322

    max q3: 1809.5

    max q4: 3954

- **The latency of your overall system for q1, q2, q3 & q4**

  latency q1: 9ms

  latency q2: 5ms

  latency q3: 5ms

  latency q1: 4ms

➢ **The cost per hour of your system at low and high load**

The high bound and low bound cost of our system is 1.2$/h. The reason why we don't use a ASG is mentioned in Chapter 3.1.

3.5 Step 5: Optimize for Throughput and Latency

➢ **The insight from Step 4 to influence optimizations**

Based on the performance in the online test, we analyze our architecture from 3 perspectives: MySQL memory, Tomcat Server and Tomcat memory and do the optimization as below.

➢ **The optimizations utilized and justifications**

✓ **MySQLInodb**

We set the innodb_buffer_pool_size = 4G, and warm up the MySQL database before the test. However, the test case's miss rate is very high, so the improvement is not obvious.

✓ **CacheMemory:**

We designed a data structure to store the data of q3, q4 in CacheMemory in front end, qps improves significantly. However, we didn't use this method in live test because it violate the purpose of this extra project and it is not fair for other teams.

✓ **LightWeight Server:**

We tried to user lightweight server such as NGINX, whose initialization is more efficient than Tomcat.

✓ **JDBC Connection Pool:**

We also implemented the connection pool to maintain a number of established connections to MySQL server.

➢ **Changes to the overall system design and implementation**

We change the JDBC connection pool and take use of the MySQL Innodb and front end's cache memory.

- **QPS and Latency**

	q1	q2	q3	q4
Throughput(qps)	5397.7	3322.5	1809.5	3954.3
Latency(ms)	9	5	5	4

- **The cost per hour of your optimized system at low and high load**

  Because we MySQL database, we combine the front and the back end. So we implement the whole system with 1 ELB and 9 m1.medium instances. The front end and back end share the same instances. We just make sure the total cost is less than 1.2 $ per hour.

  Cost:0.025 + 0.12*9 + 5 * 0.008 = 1.145<1.2

3.6 Step 6: Provision, Load and Prepare for Live Test

- **Final configuration of each part (instance type and number)**

  ELB + 9 m1.medium instance. Please reference 3.1 for the reason we select this scheme.

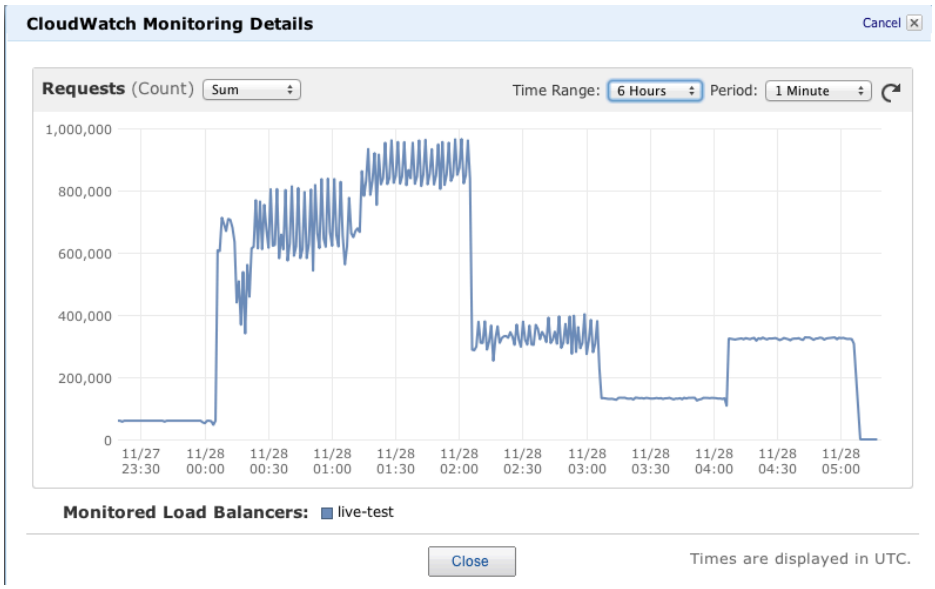
- **Estimated per hour cost of the web service during the test period**

  The ELB price is 0.025$ per hour and 0.008$ per GB of data process. The m1.medium instance price is 0.12$. The total database is about 5GB. So the total cost per hour is under:

  0.025 + 0.12*9 + 5 * 0.008 = 1.145$

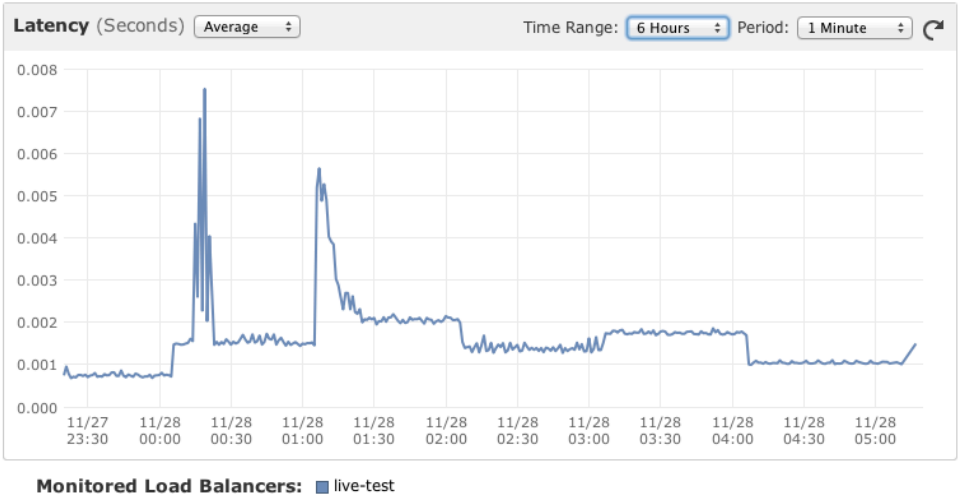
4 Live Test Performance Analysis

Please analyze your live test performance in detail and suggest possible improvement.

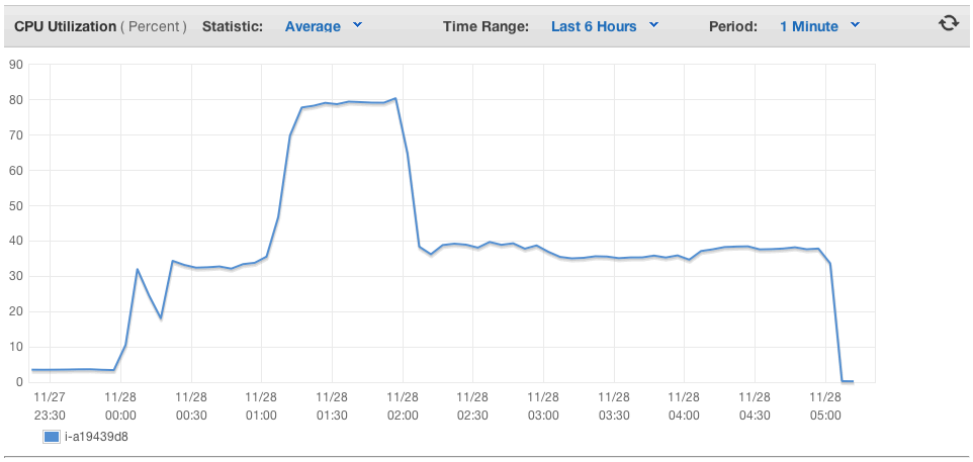


- From the Request Sum Chart above, we found that the live test is divided into 6 periods.

  Period 1: test q1 with qps around 1000
  Period 2: test q2 with qps around 5000
  Period 3: mix test with high qps
  Period 4: test q2
  Period 5: test q3
  Period 6: test q4



- For the Latency, it increase a lot when test period transfer form one to another one.



- This is the CPU Utilization of each m1.medium instance, from the data above, we can conclude that we have take full use of the CPU and it is not the bottleneck.

- **Possible Improvement**

  MySQL Innodb: We can take use Innodb of MySQL and set the innodb_buffer_pool_size = 5G, meanwhile, warm up the MySQL database before the test. However, the memory of m1.medium is not big enough to allocate to MySQL innodb as well as Tomcat memory.

  CacheMemory: The performance of q3 is the lowest, we have designed a data structure to store the data of q3 in Tomcat memory. However, we didn't use this method in live test because it violate the purpose of this extra project and it is not fair for other teams.

5 Review and Suggestions

- The size of dataset and the query types could have been designed more reasonable, so it can avoid teams to use simple database design and system architecture to pursue maximum throughput.

- The front-end design could be more challenging. We could add some UI design to make the interface more interactive and attractive to users.

- The project could encourage the teams to do some bonus, such as find some pattern in the data with data mining method and doing some visualization based on the tweets data.