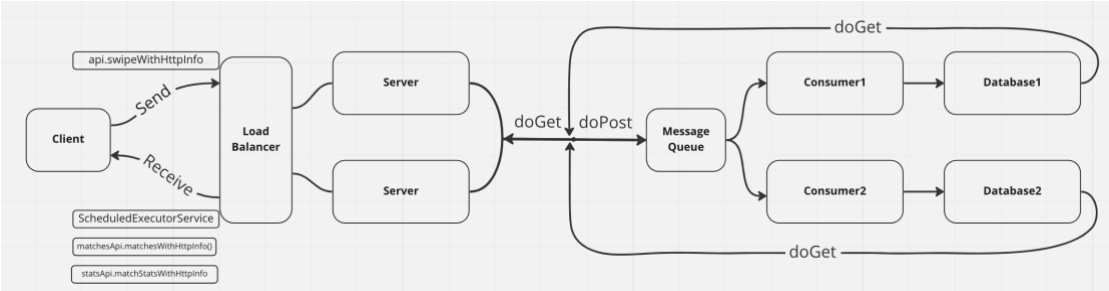
Assignment 3 Report Shengguo Zhou

URL: https://github.com/Shengguo-Zhou/CS6650_Distributed_System/tree/main

Flowchart Description:



For the assignment3, a data layer is needed. I put a Redis database behind the consumer to store the data passed from it. I implemented the doGet function inside the server. When the server will retrieve the data from database to the client. The client in assignment 3 can not only send data to server, but also can get data by calling the API from server. The details of each part is attached below:

The purpose of this report is to detail the design and implementation of a system for persisting swipe events and querying them. The solution consists of a client, a servlet, a message queue, a consumer, and a database for storing the swipe data.

Design: The solution architecture is designed to achieve high throughput and low response times while being simple and low cost. The client generates swipe events and sends them to a servlet, which writes them to a persistent queue called TempStore. A consumer reads from TempStore and updates a SwipeData database that stores information about users and swipe events. The servlet implements GET requests and retrieves results directly from the SwipeData database.

Servlet - The servlet implements doPost and writes every swipe event to TempStore. It also implements doGet and retrieves results from SwipeData.

Message Queue - TempStore is implemented as a message queue that ensures safe and persistent storage of the swipe events.

Consumer - The consumer reads from TempStore and updates Swipe Data. It ensures that the database is up to date with the latest swipe events.

Database - SwipeData is a database that stores information about users and swipe events. It is designed to enable highly efficient querying for the GET requests. In my assignment 3, I choose Redis as my database.

Client:
Main function is in “MultiThread” class. Inside this class:
First step, I created a queue and put 500k request inside it. And the queue is a global variable.

Nest step, I use a for loop to generate 200 threads to achieve multiply-thread. For each of thread, whenever this thread is started, it will pull a request from the queue. If the queue is empty, it will pull nothing, which means this thread would do nothing. So which thread is started first, which one may be tasked with more sending request tasks. Therefore, within this step, there would be 500k requests in total, and the 200 threads need to send it to the local or EC2 server. When all the requests are sent, which means the queue is empty, the 200 threads would then have a rest.

Last step, there would be some other global variable that would count the successful request number and the failed one, the lasting time and so on.

RunInOneThread.java: This class is designed for each one of the thread, I need to send the data through api.swipeWithHttpInfo. Inside the run function, the detail of each thread is implemted, queue is the global passed from the main function, when a new thread is created, it will get a request body from the queue. And it won't stop until the queue is empty. There are another two parameters to count the number of successful and failed sending tasks. In the sentEvent function, it will try at most 5 times to sent one request, if it is successful within 5 times of try, success will plus one. When it is failed, this request will be abandoned and failure will plus one.

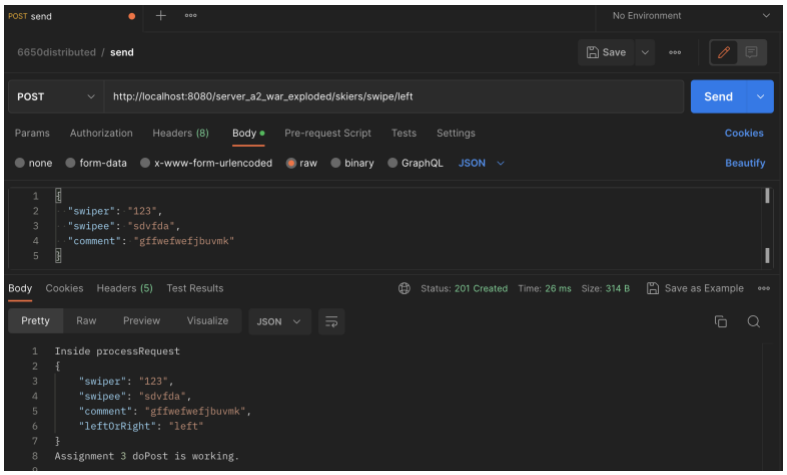
SwipeEvent.java inside the event file and BodyList.java and GenerateSingleBody.java is the helper function for main and singe-thread functions. For each of the request, I have packaged the info within the request and named it to a event class. When all the 500k request body is generated, it would be put into the list, to be specific, it is the queue that I have mentioned.

For the output part, I added nothing but result data processing class and a csv output file class. Inside the RunInOneThread.java class, when a new request is successfully sent, I will put the success info into another global list to store it. When all the requests are sent, I will get a list of 500k success info. And then I can sort this list by certain criteria. Like I can sort it by the delay time, then I can get the result of minimum, maximum, medium, 99%percentage response time. And like this, if I sort it by its start time, I can plot the chart by its beginning time to know for each of a certain second, how many requests would be sent.

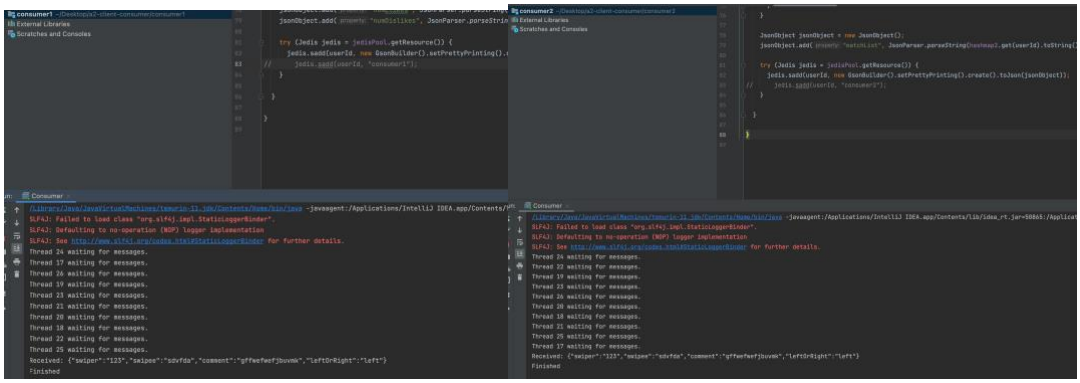
Result:
Linked the consumer with the redis database. When Postman send data to the server, the server would post it to the database, so if we get the keys inside the database, we could find the result.

I started the testing from only 1 request which is from Postman

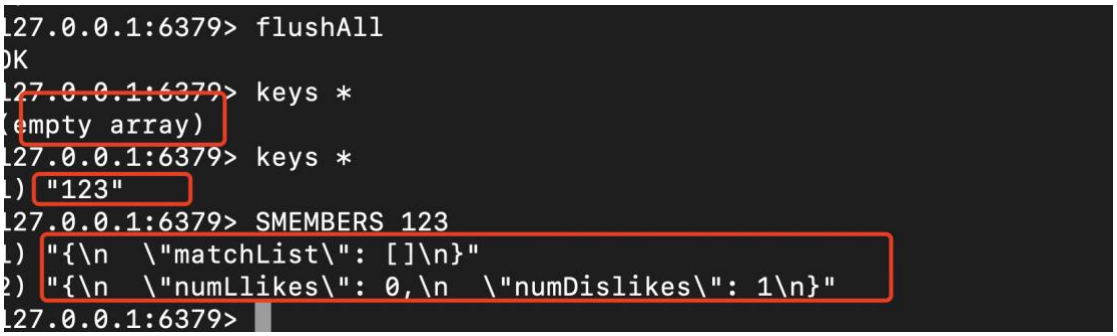
Postman:



Consumer 1 & 2 started with 10 threads for reciving messages:



Redis:

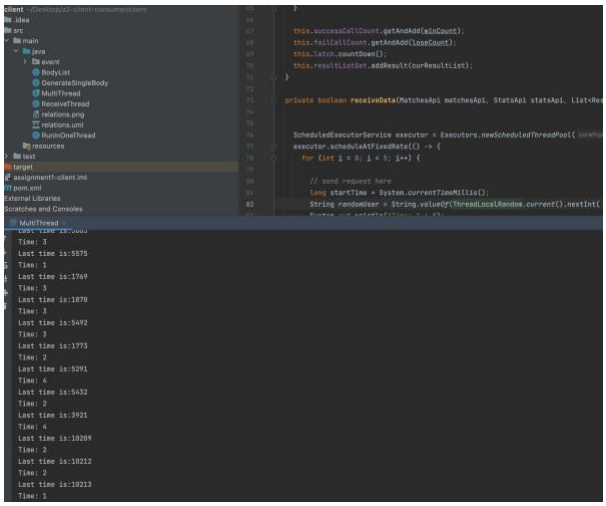


It works well, so I improved the requests number to 50.

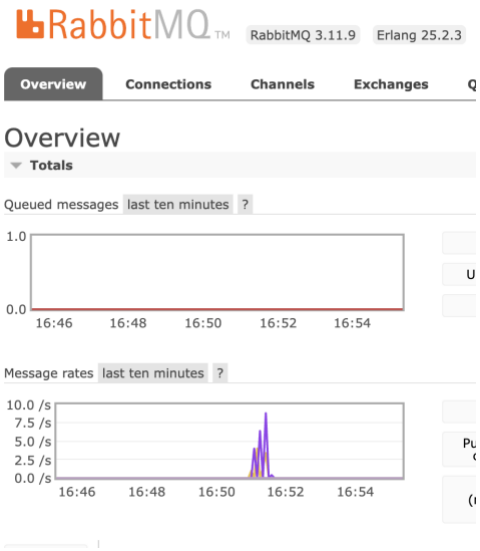
50 requests in redis:

```
35) "566"
36) "1449"
37) "3289"
38) "4643"
39) "3255"
40) "1771"
41) "3026"
42) "2340"
43) "4302"
44) "3790"
45) "3078"
46) "627"
47) "2374"
48) "338"
49) "2413"
50) "4642"
127.0.0.1:6379> SMEMBERS 4642
1) "{\n  \"matchList\": [\n    464147\n  ]\n}"
2) "{\n  \"numLlikes\": 1,\n  \"numDislikes\": 0\n}"
127.0.0.1:6379>
```

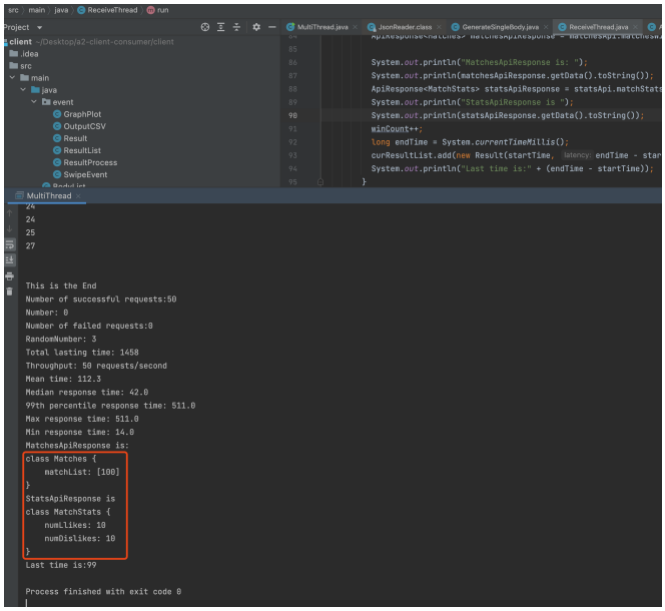
Client:



Message queue

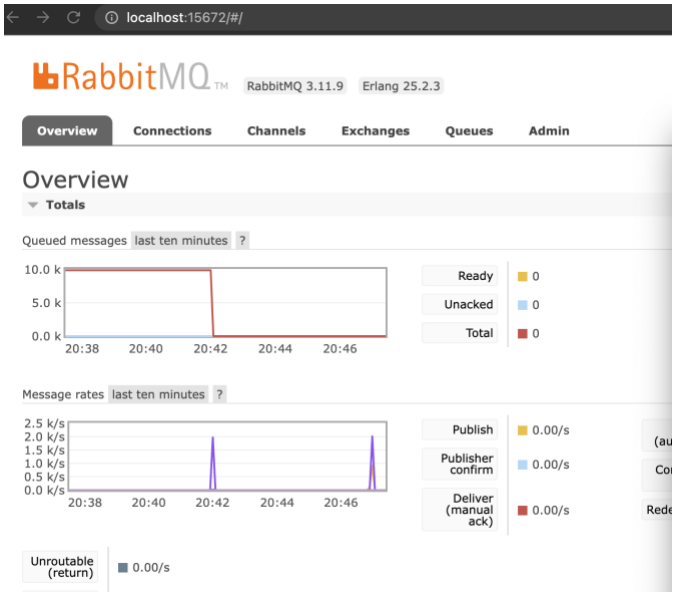


Result returned from API



It works well so then I keep on improving the number of requests:

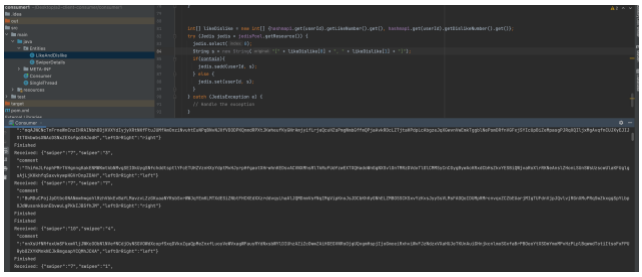
Swiper and swipee are in range of [1, 100], 5000 requests generated in total.



```
127.0.0.1:6379> keys *
 1) "4"
 2) "7"
 3) "5"
 4) "1"
 5) "8"
 6) "2"
 7) "3"
 8) "10"
 9) "9"
10) "6"
127.0.0.1:6379> get 1
"[258, 245]"
127.0.0.1:6379> select 1
OK
127.0.0.1:6379[1]> get 1
"[6, 9, 8, 4, 10, 10, 7, 4, 5, 10, 9, 7, 2, 8, 1, 4, 9, 4, 5, 4, 7, 5, 6, 5, 9,
10, 6, 9, 10, 6, 2, 3, 8, 1, 9, 4, 6, 2, 3, 1, 3, 3, 1, 4, 5, 5, 5, 2, 3, 3, 7,
3, 3, 2, 3, 1, 1, 1, 3, 6, 9, 9, 9, 8, 1, 1, 1, 5, 8, 2, 2, 2, 7, 8, 6, 3, 2, 8
 2, 8, 1, 9, 7, 4, 7, 6, 4, 8, 2, 9, 2, 1, 2, 6, 5, 3, 2, 2, 9, 8]"
127.0.0.1:6379[1]>
```

```
This is the End
Number of successful requests:5000
Number of failed requests:0
Total lasting time: 26541
Throughput: 192 requests/second
Mean time: 4.5466
Median response time: 2.0
99th percentile response time: 31.0
Max response time: 605.0
Min response time: 1.0


This is the End in receiving data
Number of successful requests in receiving data:105
Number of failed requests in receiving data:0
Total lasting time in receiving data: 26541
Throughput in receiving data: 4 requests/second
Mean time in receiving data: 3.8095238095238093
Median response time in receiving data: 2.0
99th percentile response time in receiving data: 21.0
Max response time in receiving data: 71.0
Min response time in receiving data: 0.0

Process finished with exit code 0
```

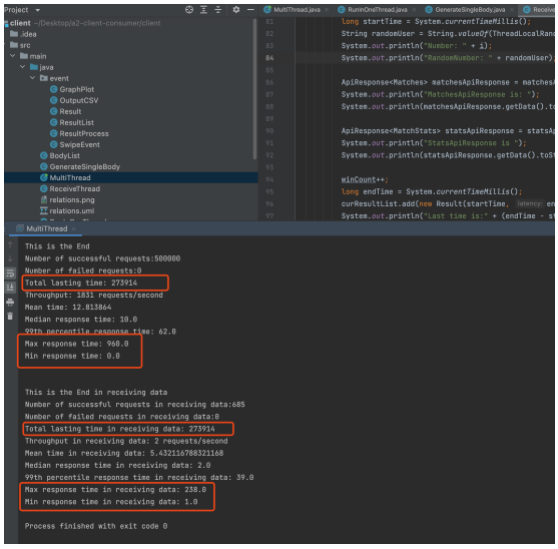
```
RandomNumber: 2
MatchesApiResponse is:
class Matches {
    matchList: [4, 3, 9, 5, 8, 9, 10, 3, 2, 2, 4, 10, 10, 1, 6, 8, 2, 10, 8, 5, 5, 10, 7, 9, 4, 5, 5, 2, 3, 8, 7, 1, 6, 1, 7, 1, 1, 9, 7,
     7, 6, 5, 8, 10, 9, 6, 3, 2, 1, 9, 3, 4, 5, 6, 5, 10, 1, 9, 6, 5, 10, 4, 10, 5, 3, 5, 2, 8, 10, 2]
}
StatsApiResponse is
class MatchStats {
    numLikes: 241
    numDislikes: 286
}
Last time is:6
Number: 4
RandomNumber: 1
MatchesApiResponse is:
class Matches {
    matchList: [6, 9, 8, 4, 10, 10, 7, 4, 5, 10, 9, 7, 2, 8, 1, 4, 9, 4, 5, 4, 7, 5, 6, 5, 9, 10, 6, 9, 10, 6, 2, 3, 8, 1, 9, 4, 6, 2, 3,
      2, 2, 7, 8, 6, 3, 2, 8, 2, 8, 1, 9, 7, 4, 7, 6, 4, 8, 2, 9, 2, 1, 2, 6, 5, 3, 2, 2, 9, 8]
}
StatsApiResponse is
class MatchStats {
    numLikes: 245
    numDislikes: 258
}
Last time is:8
```



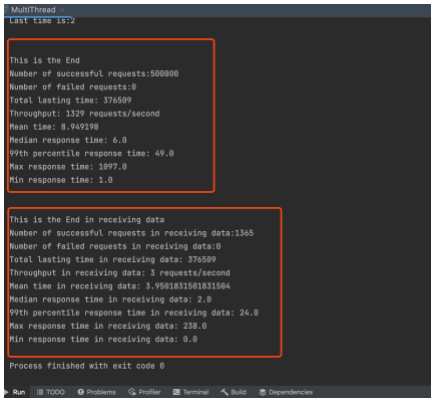
It works well, then begin our test from Local side:

1.
500k requests,
swiper & swipee range from [1, 50000],
consumer 1 & 2 : 100 threads,
client: 50threads

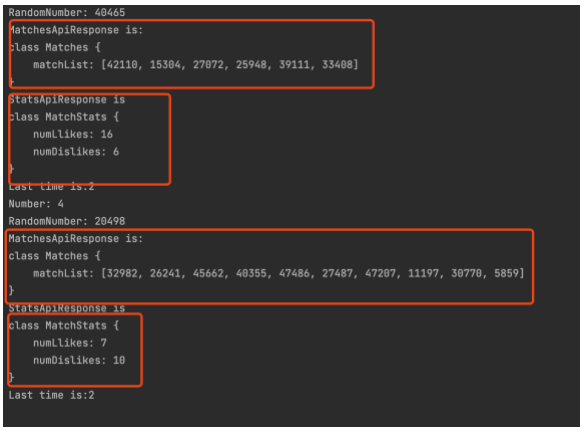
Test 1



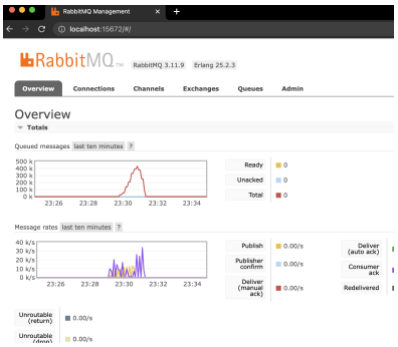
Test2



Returned value from API and Response time:



Rabbitmq



Consumer

Database:

Table 0:

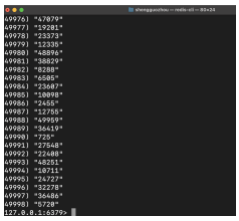
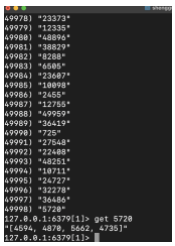


Table 1:



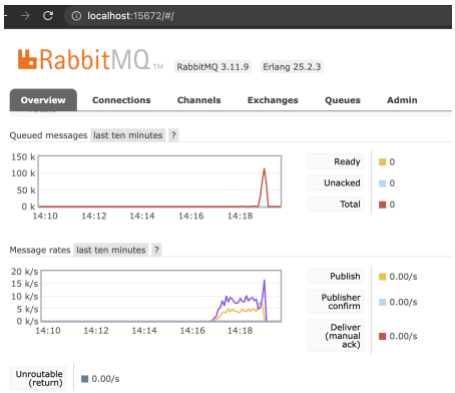
2.

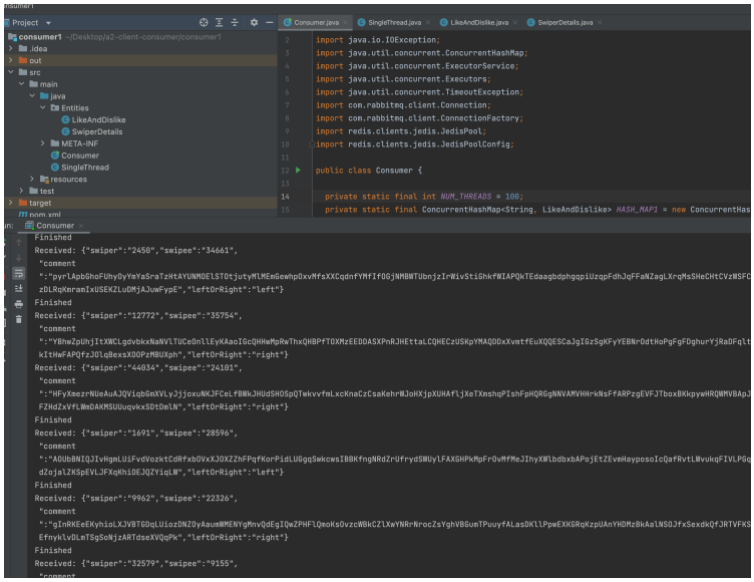
500k requests,

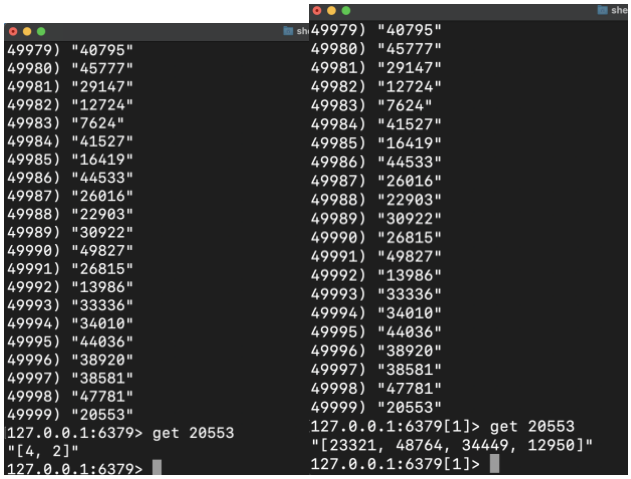
swiper & swipee range from[1, 50000],

consumer 1 & 2 : 100 threads,

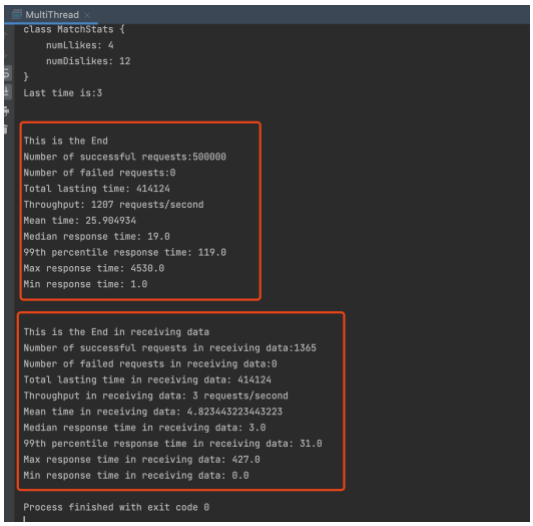
client: 100 threads

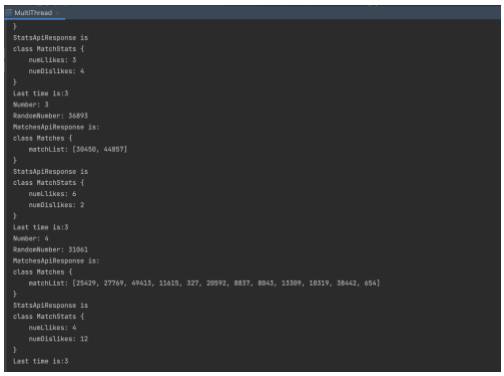

localhost:15672/#/
RabbitMQ
RabbitMQ 3.11.9
Erlang 25.2.3
Overview
Connections
Channels
Exchanges
Queues
Admin
Queued messages last ten minutes ?
150 k
100 k
50 k
0 k
14:10
14:12
14:14
14:16
14:18
Ready 0
Unacked 0
Total 0
Message rates last ten minutes ?
20 k/s
15 k/s
10 k/s
5 k/s
0 k/s
Publish 0.00/s
Publisher confirm 0.00/s
Deliver (manual ack) 0.00/s
Unroutable (return) 0.00/s


Project
consumer1 ~/Desktop/a2-client-consumer/consumer1
.idea
out
src
main
java
Entities
LikeAndDislike
SwiperDetails
META-INF
Consumer
SingleThread
resources
test
target
pom.xml
Consumer.java
SingleThread.java
LikeAndDislike.java
SwiperDetails.java
import java.io.IOException;
import java.util.concurrent.ConcurrentHashMap;
import java.util.concurrent.ExecutorService;
import java.util.concurrent.Executors;
import java.util.concurrent.TimeoutException;
import com.rabbitmq.client.Connection;
import com.rabbitmq.client.ConnectionFactory;
import redis.clients.jedis.JedisPool;
import redis.clients.jedis.JedisPoolConfig;
public class Consumer {
private static final int NUM_THREADS = 100;
private static final ConcurrentHashMap<String, LikeAndDislike> HASH_MAP1 = new ConcurrentHas
Finished
Received: {"swiper":"2450","swipee":"34661",
"comment
":"pyrlApbGhoFUhyGyYmYaSraTzHtAYUNMDElSTOtjutyMLMEmGewhpOxvMfsXXCqdnfYMfIfOGjNMBWTUbnjzIrWivStiGhkfWIAPQkTEdaagbdphgqpiUzqpFdhJqFFaNZagLXrqMsSHeCHtCVzWSFC
zDLRqKmramIxUSEKZLuDMjAJuwFypE","leftOrRight":"left"}
Finished
Received: {"swiper":"12772","swipee":"35754",
"comment
":"YBhwZpUhjItXWCLgdvbkxNaNVlTUCeOnllEyKAaoIGcQHHwMpRwThxQHBPfTQXMzEEDDASXPnRJHEttaLCQHECzUSKpYMAQODxXvmtfEoXQQESCaJgIGzSgKFyYEBNrOdtHoPgFgFDghurYjRaDFqlt
kItHwFAPQfzJDlqBexsXOOPzMBUXph","leftOrRight":"right"}
Received: {"swiper":"44034","swipee":"24101",
"comment
":"HFyXmezrNUeAuAJQViqbGmXVLyJjjoxuNKJFCeLfBWkJHUdSHOSpQTwkvvfmLxcKnaCzCsaKehrWJoHXjpXUHAfljXeTXmshqPIshFpHQRGgNNVAMVHHrkNsFfARPzgEVFJTboxBKkpywHRQNMVBApJ
FZHdZxVfLWmDAKMSUUuqvkxSDtOmlN","leftOrRight":"right"}
Finished
Received: {"swiper":"1691","swipee":"28596",
"comment
":"AOUb8NIQJIvHgmLUiFvdVozktCdRfxbOVxXJOXZZhFPqfKorPidLUGgqSwkcwsIBBKfngNRdZrUfrydSWUylFAXGHPkMpFrOvMfMeJIhyXWlbdbxbAPojEtZEveHayposoIcQafRvtLWvukqFIVLPGq
dZojalZKSpEVLJFXqKhiDEJQZYiqLW","leftOrRight":"left"}
Finished
Received: {"swiper":"9962","swipee":"22326",
"comment
":"gIrNKEeEKyhioLXJVBTGDqLUiozDNZOyAauwWMEKYgMnvQdEgIQwZPHFlQmoKsOvzcWbKCZlXwYNRrNrocZsYghVBGumTPuuyfALasDKllPgwEXKSRqKzpUAnYHDMzBkAalNSDJfxSexdkQfJRTVFKS
EfnykIvDLmTSgSoNjzARTdseXVQqPk","leftOrRight":"right"}
Finished
Received: {"swiper":"32579","swipee":"9155",
"comment


49979) "40795"
49980) "45777"
49981) "29147"
49982) "12724"
49983) "7624"
49984) "41527"
49985) "16419"
49986) "44533"
49987) "26016"
49988) "22903"
49989) "30922"
49990) "49827"
49991) "26815"
49992) "13986"
49993) "33336"
49994) "34010"
49995) "44036"
49996) "38920"
49997) "38581"
49998) "47781"
49999) "20553"
127.0.0.1:6379> get 20553
"[4, 2]"
127.0.0.1:6379>

49979) "40795"
49980) "45777"
49981) "29147"
49982) "12724"
49983) "7624"
49984) "41527"
49985) "16419"
49986) "44533"
49987) "26016"
49988) "22903"
49989) "30922"
49990) "26815"
49991) "49827"
49992) "13986"
49993) "33336"
49994) "34010"
49995) "44036"
49996) "38920"
49997) "38581"
49998) "47781"
49999) "20553"
127.0.0.1:6379[1]> get 20553
"[23321, 48764, 34449, 12950]"
127.0.0.1:6379[1]>





3.

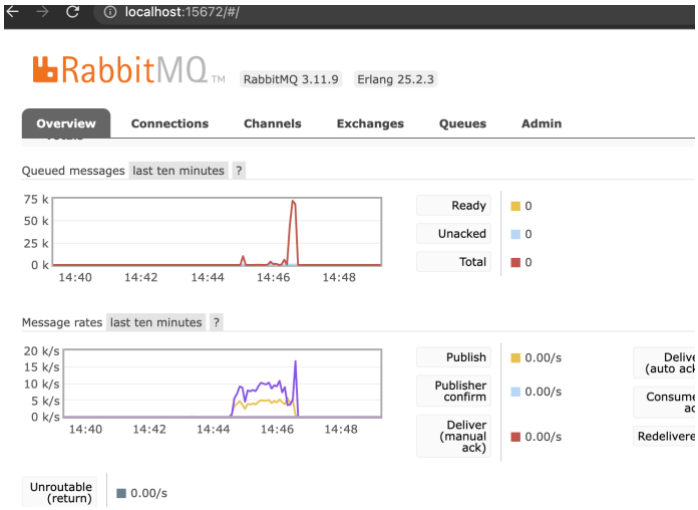
500k requests,

swiper & swipee range from[1, 50000],

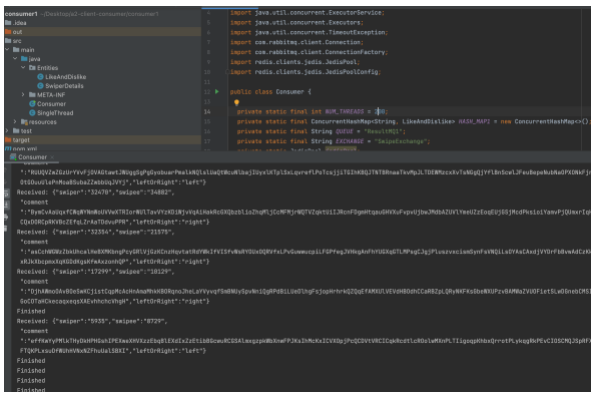
consumer 1 & 2 : 200 threads,

client: 200 threads

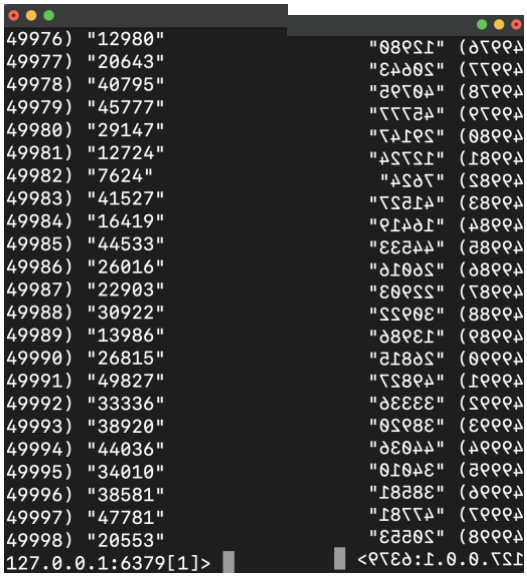
Rabbitmq



consumer

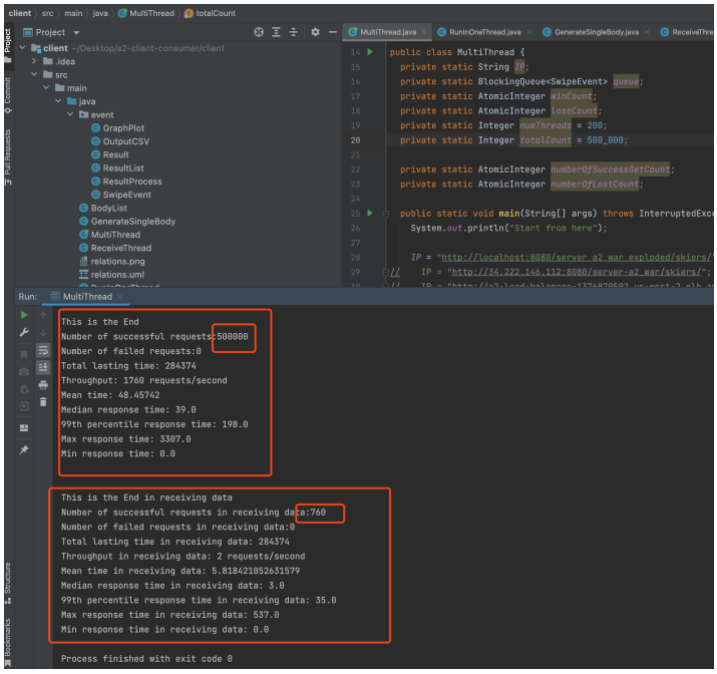


Redis Table 0 and 1



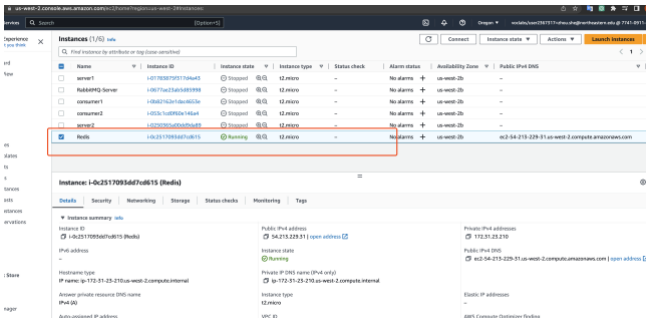
Client

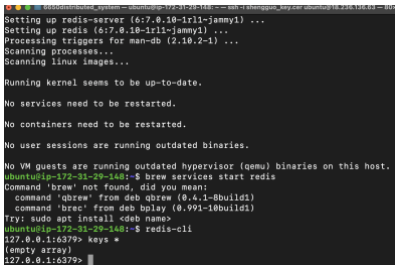
```
StatsApiResponse is
class MatchStats {
    numLlikes: 4
    numDislikes: 6
}
Last time is:1
Number: 3
RandomNumber: 20667
MatchesApiResponse is:
class Matches {
    matchList: [27642, 27355, 14771, 23683, 30922, 29695]
}
StatsApiResponse is
class MatchStats {
    numLlikes: 9
    numDislikes: 6
}
Last time is:1
Number: 4
RandomNumber: 43503
MatchesApiResponse is:
class Matches {
    matchList: [5002, 48220, 41032]
}
StatsApiResponse is
class MatchStats {
    numLlikes: 6
    numDislikes: 3
}
Last time is:2
```



The following part are tests from AWS side:

A new instance for Redis and give the access to it.





Ip address:

Database redis: 18.236.136.63

Consumer1: 52.11.54.7

Consumer2: 54.213.114.185

Rabbitmq: 35.166.166.121

Server1:35.166.157.115

Server2: 54.212.9.5

loadBalancer: a3-lb-709645041.us-west-2.elb.amazonaws.com

Database redis: 34.213.111.105

Consumer1: 54.202.50.200

Consumer2: 54.186.31.145

Rabbitmq: 18.237.80.111

Server1: 35.86.147.140

Server2: 34.222.61.141

loadBalancer: a3-lb-709645041.us-west-2.elb.amazonaws.com

Database redis: 34.219.132.175

Consumer1: 54.200.33.85

Consumer2: 54.202.70.33

Rabbitmq: 54.201.155.64

Server1: 35.92.44.101

Server2: 52.10.99.67

loadBalancer: a3-lb-709645041.us-west-2.elb.amazonaws.com

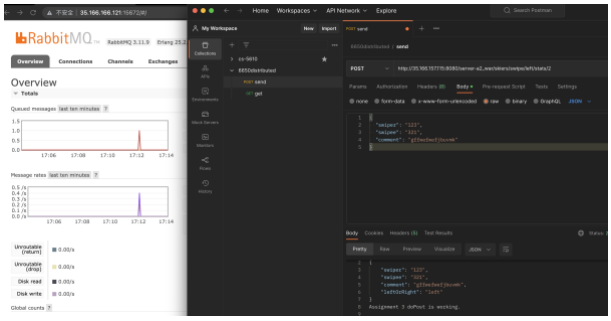
```
[ec2-user@ip-172-31-31-213 ~]$ ls
consumer2.jar
[ec2-user@ip-172-31-31-213 ~]$ java -jar consumer2.jar
SLF4J: Failed to load class "org.slf4j.impl.StaticLoggerBinder".
SLF4J: Defaulting to no-operation (NOP) logger implementation
SLF4J: See http://www.slf4j.org/codes.html#StaticLoggerBinder for further det
s.
Thread 67 waiting for messages.
Thread 117 waiting for messages.
Thread 163 waiting for messages.
Thread 116 waiting for messages.
Thread 115 waiting for messages.
Thread 114 waiting for messages.
Thread 113 waiting for messages.
Thread 112 waiting for messages.
```

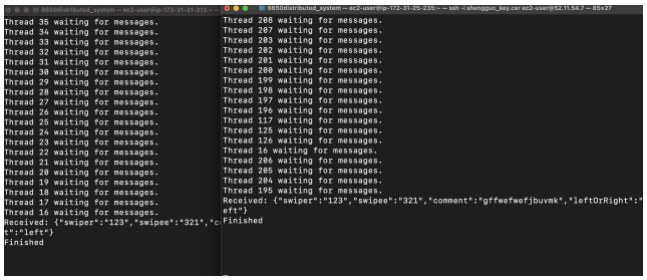
```
6650distributed_system — ec2-user@ip-172-31-25-235:~ — ssh -i shengguo_key.cer ec2-user@52.11.54.7 — 80×24
-bash: warning: setlocale: LC_CTYPE: cannot change locale (UTF-8): No such file
or directory
[ec2-user@ip-172-31-25-235 ~]$ ls
consumer1.jar
[ec2-user@ip-172-31-25-235 ~]$ java -jar consumer1.jar
SLF4J: Failed to load class "org.slf4j.impl.StaticLoggerBinder".
SLF4J: Defaulting to no-operation (NOP) logger implementation
SLF4J: See http://www.slf4j.org/codes.html#StaticLoggerBinder for further detai
s.
Thread 22 waiting for messages.
Thread 140 waiting for messages.
Thread 139 waiting for messages.
Thread 138 waiting for messages.
Thread 137 waiting for messages.
Thread 193 waiting for messages.
Thread 127 waiting for messages.
Thread 125 waiting for messages.
```

Pass the .war file to server (name is unchanged)

```
ec2-user@ip-172-31-21-39 tomcat]$ ls
UILDING.txt     LICENSE  README.md      RUNNING.txt  conf  logs  webapps
ONTRIBUTING.md  NOTICE   RELEASE-NOTES  bin          lib   temp  work
ec2-user@ip-172-31-21-39 tomcat]$ cd webapps/
ec2-user@ip-172-31-21-39 webapps]$ ls
OOT  docs  examples  host-manager  manager  server-a2_war.war
ec2-user@ip-172-31-21-39 webapps]$ ls
OOT  docs  examples  host-manager  manager  server-a2_war  server-a2_war.war
ec2-user@ip-172-31-21-39 webapps]$
```

1 request from Postman





1.

500k requests,

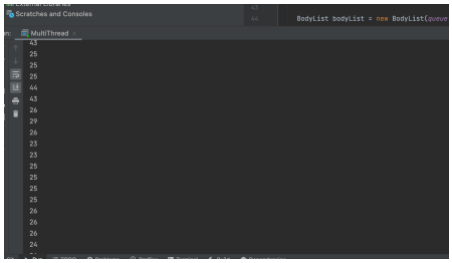
swiper & swipee range from[1, 50000],

consumer 1 & 2 : 200 threads,

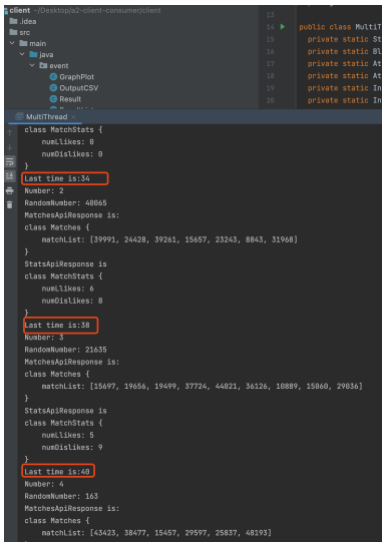
client: 50 threads

Clinet:

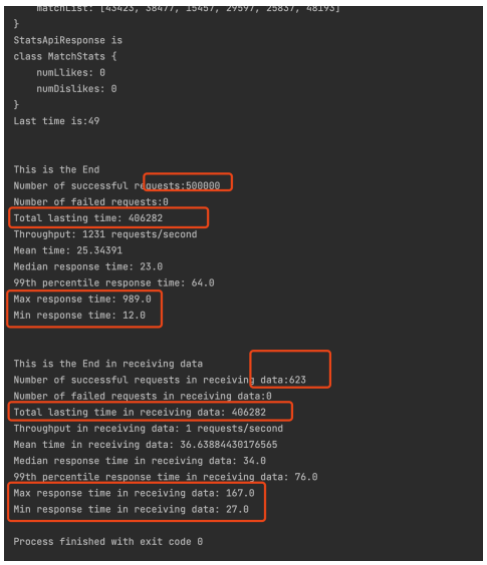
Sending data's delay time is around 20ms – 40ms



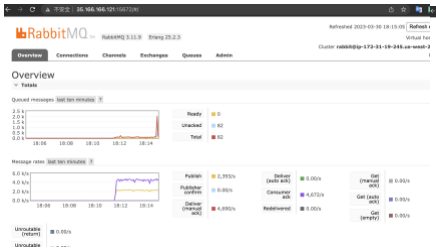
Receiving delay time and the result got after calling the two required API

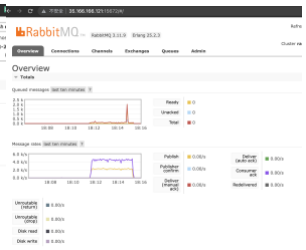


Final result:



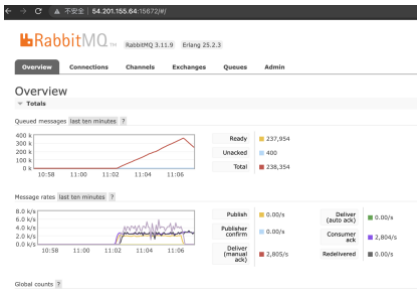
rabbitmq



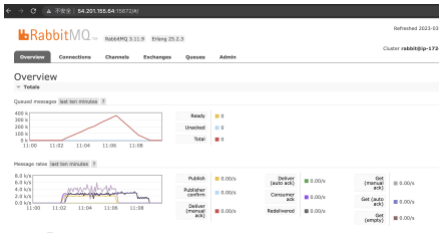


Test 2

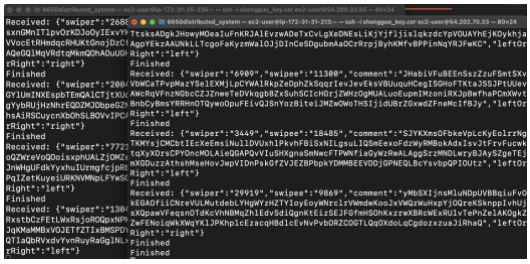
Turning point is shown:



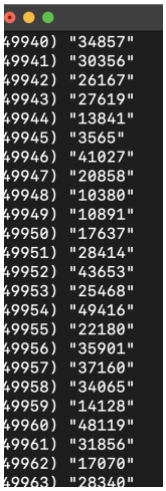
Final:



Consumer



Redis



```
ubuntu@ip-172-31-29-148:~$ redis-cli
127.0.0.1:6379> keys *
(empty array)
127.0.0.1:6379> select 1
OK
```

```
6650distributed_system — ubuntu@ip-172-31-29
49913) "44091"
49914) "7002"
49915) "42325"
49916) "48786"
49917) "32586"
49918) "26065"
49919) "8752"
49920) "24138"
49921) "20130"
49922) "20799"
49923) "7708"
49924) "21274"
49925) "34391"
49926) "35255"
49927) "34138"
49928) "47468"
49929) "39423"
49930) "22536"
49931) "49927"
49932) "8987"
49933) "7039"
49934) "3327"
(0.57s)
127.0.0.1:6379[1]>
```

2.

500k requests,

swiper & swipee range from[1, 50000],

consumer 1 & 2 : 200 threads,

client: 100 threads

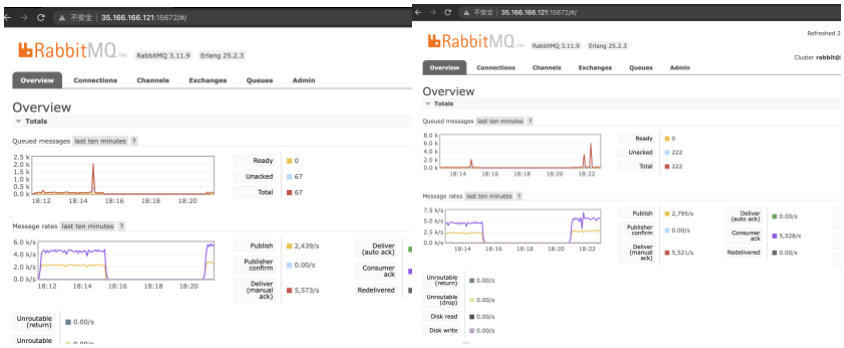
client:

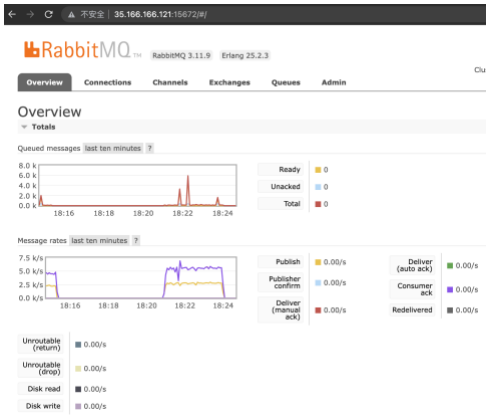
```
BodyList
GenerateSingleBody
MultiThread
    numLlikes: 9
    numDislikes: 17
}
Last time is:36
Number: 2
RandomNumber: 4354
MatchesApiResponse is:
class Matches {
    matchList: [45172, 26415, 6531, 49520, 170, 6955, 31544]
}
StatsApiResponse is
class MatchStats {
    numLlikes: 12
    numDislikes: 9
}
Last time is:32
Number: 3
RandomNumber: 23421
MatchesApiResponse is:
class Matches {
    matchList: [27939, 2480, 27780, 36614, 16128, 25230, 30970, 2024
}
StatsApiResponse is
class MatchStats {
    numLlikes: 11
    numDislikes: 11
}
Last time is:33
Number: 4
RandomNumber: 270
MatchesApiResponse is:
```

```
StatsApiResponse is
class MatchStats {
    numLikes: 14
    numDislikes: 9
}
Last time is:36
Number: 1
RandomNumber: 1019
MatchesApiResponse is:
class Matches {
    matchList: [25563, 29627, 36051, 6575, 16731, 8280, 46647, 28867, 24352, 34194, 27451, 27069, 1013, 46968, 2420, 36
}
StatsApiResponse is
class MatchStats {
    numLikes: 9
    numDislikes: 17
}
Last time is:36
```

```
BodyList
GenerateSingleBody
MultiThread
    numLikes: 10
    numDislikes: 10
}
Last time is:32


This is the End
Number of successful requests:500000
Number of failed requests:0
Total lasting time: 332274
Throughput: 1506 requests/second
Mean time: 43.921226
Median response time: 40.0
99th percentile response time: 112.0
Max response time: 1700.0
Min response time: 14.0


This is the End in receiving data
Number of successful requests in receiving data:450
Number of failed requests in receiving data:0
Total lasting time in receiving data: 332274
Throughput in receiving data: 1 requests/second
Mean time in receiving data: 39.34
Median response time in receiving data: 37.0
99th percentile response time in receiving data: 78.0
Max response time in receiving data: 157.0
Min response time in receiving data: 27.0

Process finished with exit code 0
```

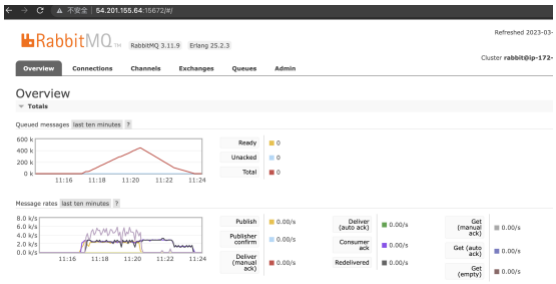
Rabbitmq





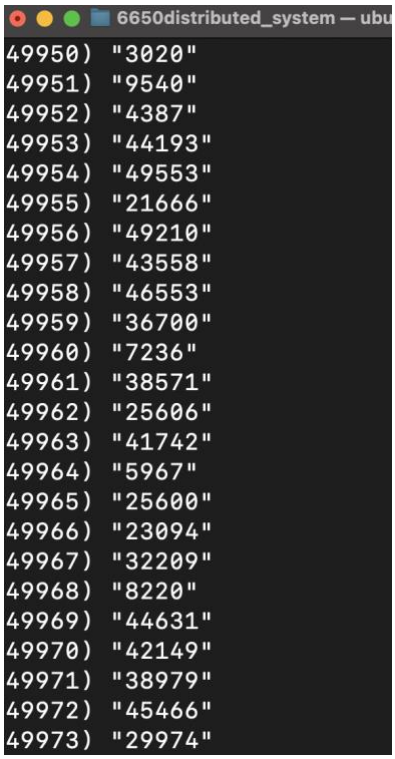
Turning point appeared:

Final:



consumer

Redis



3.

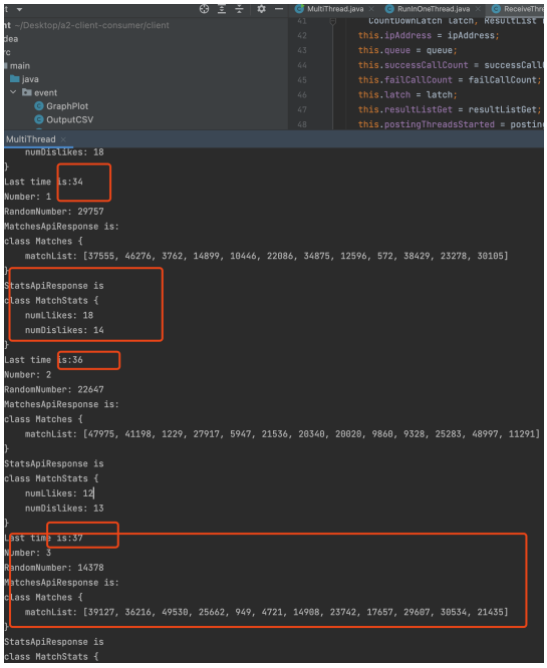
500k requests,

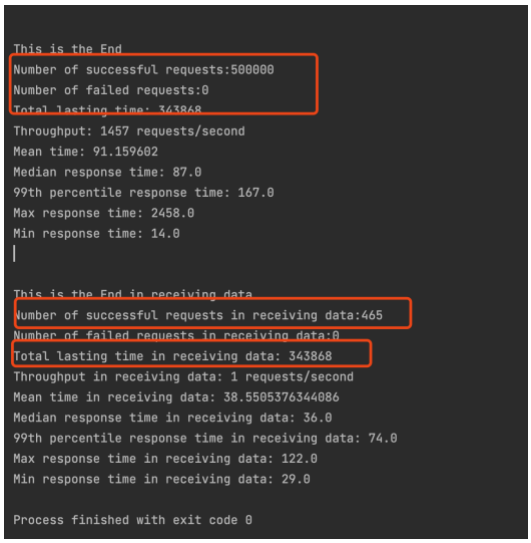
swiper & swipee range from[1, 50000],

consumer 1 & 2 : 200 threads,

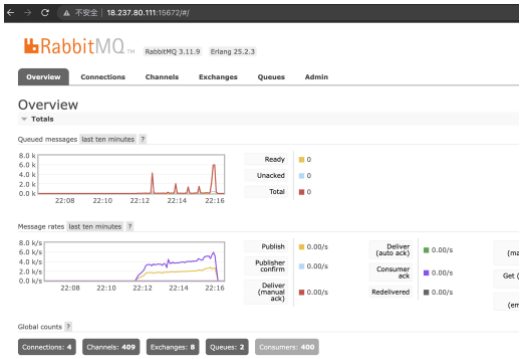
client: 200 threads

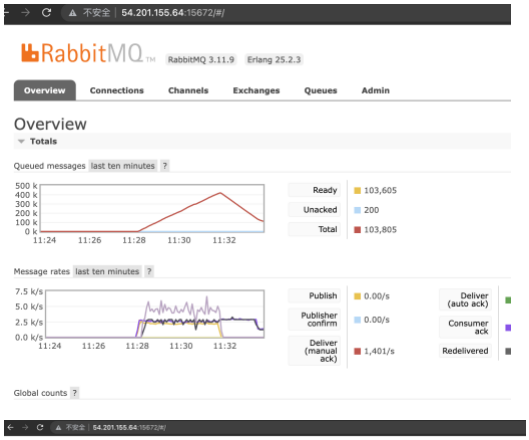
client:

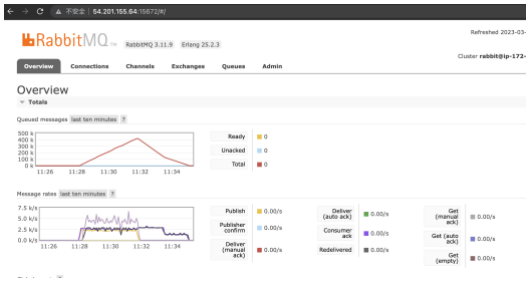




Rabbitmq







consumer

Redis

```
49976) "41979"
49977) "37599"
49978) "34598"
49979) "16461"
49980) "19176"
49981) "22450"
49982) "30952"
49983) "12604"
49984) "43143"
49985) "42269"
49986) "43807"
49987) "20057"
49988) "5991"
49989) "12612"
49990) "25110"
49991) "35276"
49992) "17516"
49993) "49799"
49994) "39004"
49995) "3984"
49996) "16987"
49997) "30260"
49998) "8457"
127.0.0.1:6379>
```

```
ubuntu@ip-172-31-29-148:~$ redis-cli ping
PONG
ubuntu@ip-172-31-29-148:~$ redis-cli
127.0.0.1:6379> keys *
(empty array)
127.0.0.1:6379> keys *
```

```
6650distributed_system — ubunt
49981) "10757"
49982) "48786"
49983) "42325"
49984) "32586"
49985) "7002"
49986) "26065"
49987) "8752"
49988) "24138"
49989) "20130"
49990) "7708"
49991) "20799"
49992) "21274"
49993) "34391"
49994) "35255"
49995) "47468"
49996) "34138"
49997) "39423"
49998) "22536"
49999) "49927"
50000) "8987"
50001) "7039"
50002) "3327"
(0.52s)
127.0.0.1:6379>
```

Each part also works well on AWS EC2 Instance

Results: My solution was able to achieve throughput and latencies at the client within *10% of Assignment 2. The GET request mean latency was also not longer than the mean POST request latency. (36ms < 78ms in medium response time among the best performance results). *with the comparision with Report 2: link

Conclusion: In conclusion, my solution provides a reliable and efficient way of persisting swipe events and querying them. By utilizing a message queue and a consumer, I was able to ensure safe and persistent storage of the swipe events while keeping latencies low. My choice of database also enabled highly efficient querying for the GET requests. Overall, I was able to achieve results that met the requirements of the assignment while keeping the architecture and deployment simple and low cost.