DOS Project 4 Part I

Yu, Mingjun (UFID 6170-7843)

Sun, Hui(UFID 6654-2614)

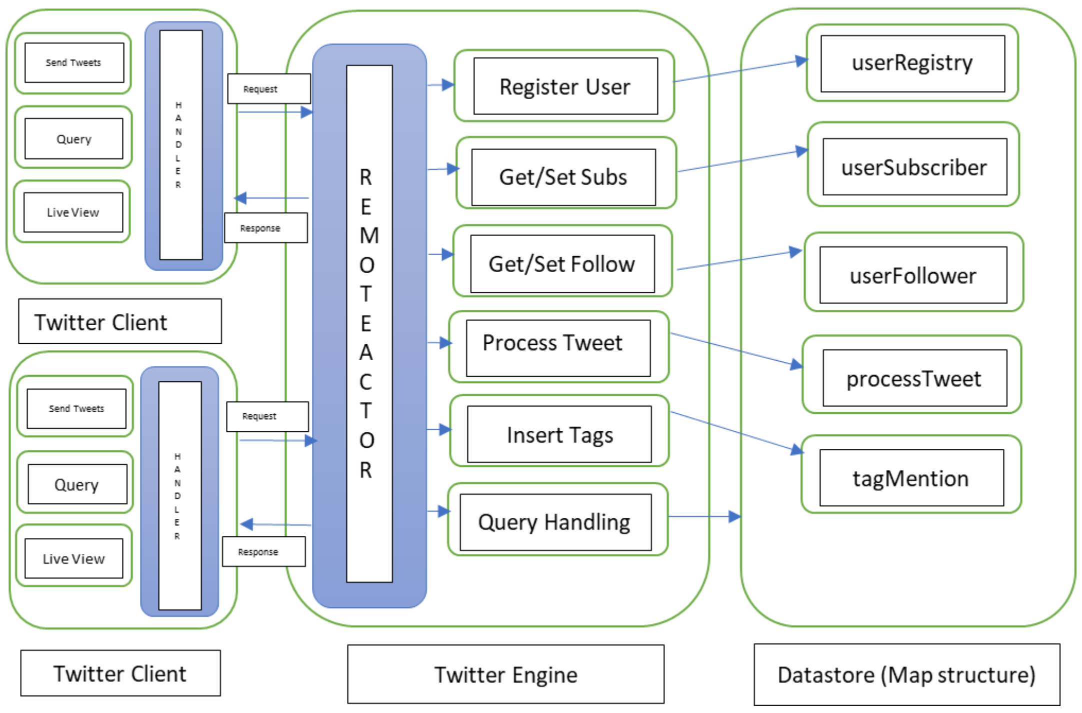
Brief Description:

The goal of this project is to implement a Twitter Clone and a client tester/simulator. The problem statement is to implement an engine that (in part II) will be paired up with WebSockets to provide full functionality. The client part (send/receive tweets) and the engine (distribute tweets) were simulated in separate OS processes. Multiple independent client processes that represented up to 10 million simulated clients were spawned during an iteration and handled by a single server process.

The Twitter engine (Server) has been implemented with the following functionality:

- Register account
- Send tweet. Tweets can have hashtags (e.g. #COP5615isgreat) and mentions (@bestuser)
- Subscribe to user's tweets
- Re-tweets (so that your subscribers get an interesting tweet you got by other means)
- Allow querying tweets subscribed to, tweets with specific hashtags, tweets in which the user is mentioned (my mentions)
- If the user is connected, deliver the above types of tweets live (without querying)

Architecture



Implementation Details:

We separately implement the server and client.

Server End:

In the server end, we have three files, Message.fs, Program.fs and Server.fs.

Program.fs: This file is the main entry of the server end. It also helps to process the communicate with the client end.

Server.fs: This file contains the code for Twitter engine implementation that is responsible for storing and processing the request from the client end. It also process the tweets from the clients so that these tweets could be stored at different “database table” which could help server find the results quickly.

Message.fs: This file provides the norm of the message transmit between sever and client.

Client End:

Program.fs: This file is the entry point of client. Our implementation takes 3 parameters in the order - numClients, maxSubscribers, disconnectClients

- numClients: the number of clients to simulate (eg. 100)
- maxSubscribers: the maximum number of subscribers a Twitter account can have in the simulation (eg. 50)
- disconnectClients: the percentage of clients to disconnect to simulate periods of live connection and disconnection (eg. 10)

Client.fs: This file consists of the information pertaining to a single client participant that stands for a single twitter user. This includes the information of its userId which is stored as a numeric string passed to it upon initiation.

Message.fs: This file provides the norm of the message transmit between server and client.

Zipf distribution - As per the requirements we were asked to simulate a Zipf distribution on the number of subscribers. The Zipf distribution was handled by taking an additional parameter from the user, maxSubscribers which provides the maximum number of subscribers a client can have in the simulation. The client with the second highest number of subscribers had maxSubscribers/2 total subscribers while the client with the third highest number of subscribers had maxSubscribers/3 total subscribers and so on. This was achieved by calculating the number of clients a user should subscribe to in order to achieve zipf distribution using the formula noToSubscribe = round(floor(totalSubscribers/(noOfClients-count+1))) – 1, where ‘count’ is the userId of a client. Every client then subscribed to noToSubscribe other clients thereby achieving the required Zipf distribution. Further, we also increased the number of tweets.

For accounts with a lot of subscribers the number of tweets had to be increased so as to increase load on the engine for tweet distribution. This was done by ensuring that for any user account, its total tweets are equal to the number of its subscribers, using the formula round(floor(totalSubscribers/count)) for calculating noOfTweets.

Periods of live connection and disconnection for users – The third parameter taken by the program is disconnect clients which takes in the percentage of clients to disconnect to simulate periods of live connection and disconnection. If <disconnectClients> parameter is 0, the client’s simulator console displays the performance statistics at the end. Otherwise, it prints the statistics and continues to simulate recurring periods of live connection and disconnection.

Re-tweets

Re-tweets are handled by making every client randomly pick a tweet from one of its subscribers and retweet it to its own subscribers. A “-RT” is appended to the end of a retweet to differentiate it from normal tweets as in original Twitter.

All the queries with tweets subscribed to, tweets with specific hashtags, tweets in which the user is mentioned (my mentions) were handled successfully by message passing to the server and displaying the list of tweets received on the client side. The tweets will be delivered live to a user that is online by means of live view. User ID is prefixed to this output to identify which user's live view is getting updated.

How to Run:

In order to run our program, we need some environment requires below:

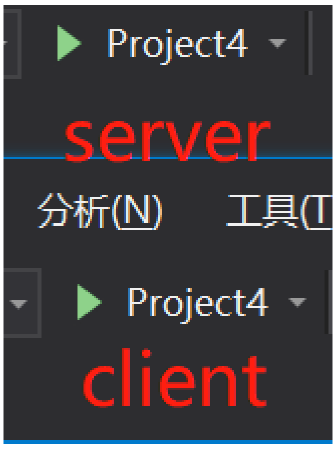
[1] Windows Operating System.

[2] Visual Studio 2019.

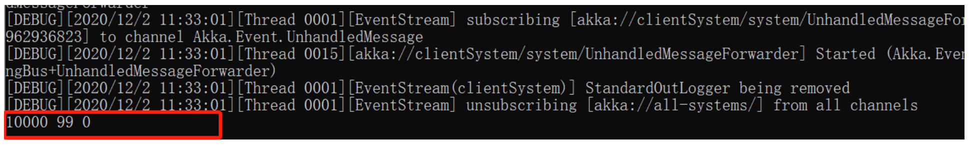
[3] F# and Akka.

[4] Firstly, open the Project4.sln under Project4Server to load server end. Then, open the Project4.sln uder Project4Client to load client end.

[5] Click ‘run’ button of server to run server, and then click ‘run’ button of client to run client like below.



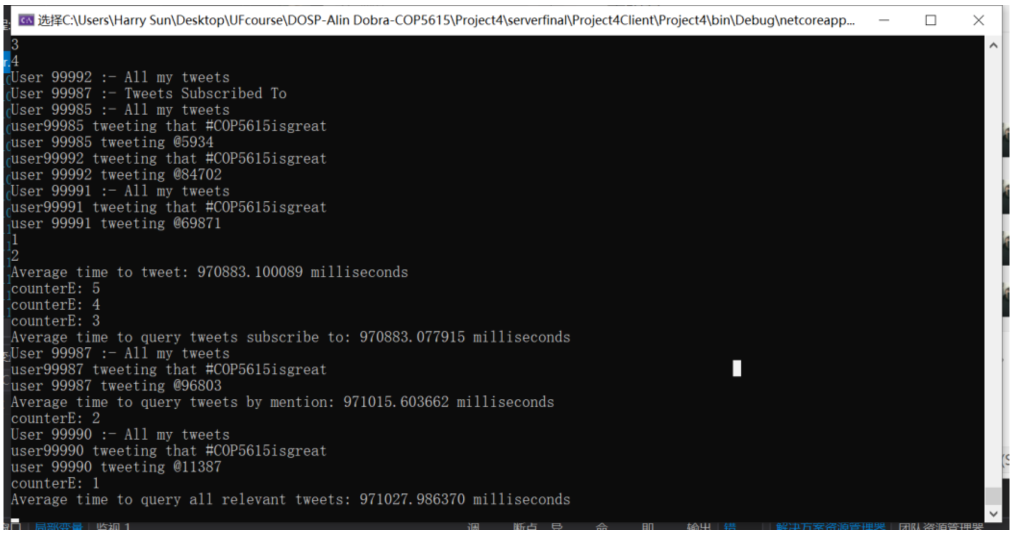
[6] Then input parameters as required in the window of client like below.



```
[DEBUG][2020/12/2 11:33:01][Thread 0001][EventStream] subscribing [akka://clientSystem/system/UnhandledMessageFor
962936823] to channel Akka.Event.UnhandledMessage
[DEBUG][2020/12/2 11:33:01][Thread 0015][akka://clientSystem/system/UnhandledMessageForwarder] Started (Akka.Even
ngBus+UnhandledMessageForwarder)
[DEBUG][2020/12/2 11:33:01][Thread 0001][EventStream(clientSystem)] StandardOutLogger being removed
[DEBUG][2020/12/2 11:33:01][Thread 0001][EventStream] unsubscribing [akka://all-systems/] from all channels
10000 99 0
```

Performance Results:

The time taken for the simulation to run vs the number of clients is plotted as a graph. A maximum of **100,000 clients** have been spawned on our system.



选择C:\Users\Harry Sun\Desktop\UFcourse\DOSP-Alin Dobra-COP5615\Project4\serverfinal\Project4Client\Project4\bin\Debug\netcoreapp...

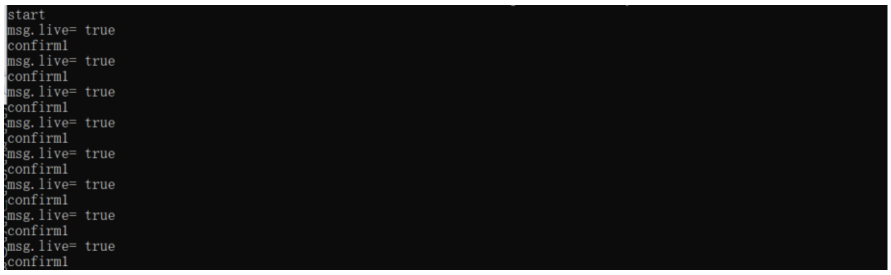
```
3
4
User 99992 :- All my tweets
User 99987 :- Tweets Subscribed To
User 99985 :- All my tweets
user99985 tweeting that #COP5615isgreat
user 99985 tweeting @5934
user99992 tweeting that #COP5615isgreat
user 99992 tweeting @84702
User 99991 :- All my tweets
user99991 tweeting that #COP5615isgreat
user 99991 tweeting @69871
1
2
Average time to tweet: 970883.100089 milliseconds
counterE: 5
counterE: 4
counterE: 3
Average time to query tweets subscribe to: 970883.077915 milliseconds
User 99987 :- All my tweets
user99987 tweeting that #COP5615isgreat
user 99987 tweeting @96803
Average time to query tweets by mention: 971015.603662 milliseconds
counterE: 2
User 99990 :- All my tweets
user99990 tweeting that #COP5615isgreat
user 99990 tweeting @11387
counterE: 1
Average time to query all relevant tweets: 971027.986370 milliseconds
```

When the number of clients is **100,000**, max subscriber is 99 and disconnect ration is 0, from the screen shot above, we could see that the average time to query all relevant tweets is **971027.986370 milliseconds**.

Since our client server implementation is completely distributed, we are confident that more clients can be spawned with suitable availability of CPU power.

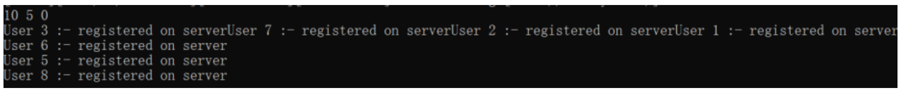
Here we use the input "10 5 0" to illustrate our program.

[1] When we click run button of server's program.fs. We could see "start" which means the server run successfully.



```
start
msg.live= true
confirml
msg.live= true
confirml
msg.live= true
confirml
msg.live= true
confirml
msg.live= true
confirml
msg.live= true
confirml
msg.live= true
confirml
msg.live= true
confirml
```

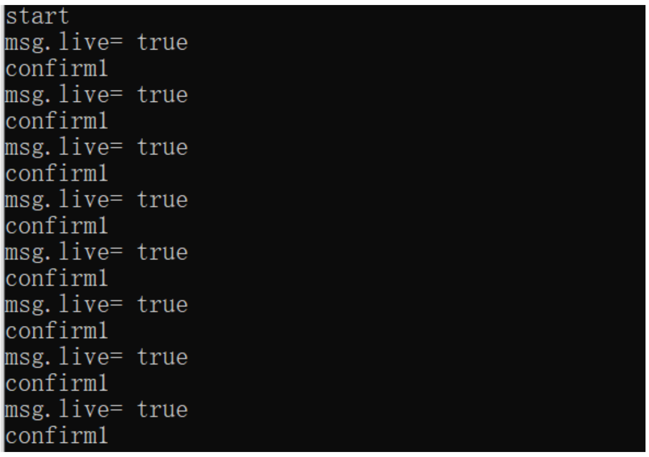
[2] Then after we input 10 5 0, the output information shows below



```
10 5 0
User 3 :- registered on serverUser 7 :- registered on serverUser 2 :- registered on serverUser 1 :- registered on server
User 6 :- registered on server
User 5 :- registered on server
User 8 :- registered on server
```

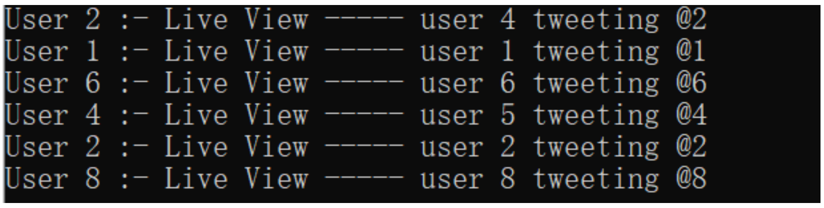
The information likes "User 3 :- registered on serverUser" means the account 3 **has already registered.**

[3] Then the server would output confirm to show that this account **has already registered.**



```
start
msg.live= true
confirm1
msg.live= true
confirm1
msg.live= true
confirm1
msg.live= true
confirm1
msg.live= true
confirm1
msg.live= true
confirm1
msg.live= true
confirm1
msg.live= true
confirm1
```

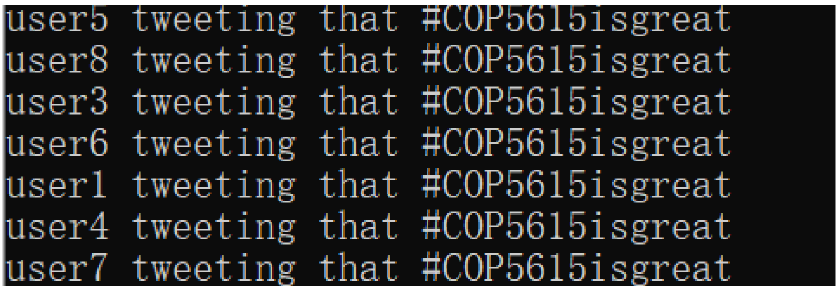
[4] The client would output information once the specific user **was mentioned**. If this mentioned user is online, then it would receive this tweet.



```
User 2 :- Live View ----- user 4 tweeting @2
User 1 :- Live View ----- user 1 tweeting @1
User 6 :- Live View ----- user 6 tweeting @6
User 4 :- Live View ----- user 5 tweeting @4
User 2 :- Live View ----- user 2 tweeting @2
User 8 :- Live View ----- user 8 tweeting @8
```

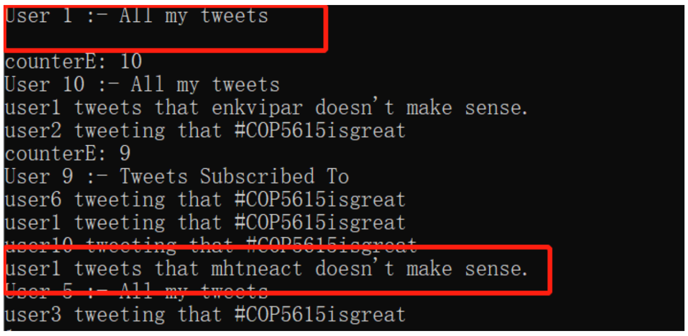
Above shows that the user 2,1,6,4,2,8 would see the tweets which mentioned them.

[5] Also, in the client, some users would **publish some tweets with hashtag**



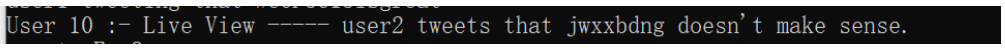
```
user5 tweeting that #COP5615isgreat
user8 tweeting that #COP5615isgreat
user3 tweeting that #COP5615isgreat
user6 tweeting that #COP5615isgreat
user1 tweeting that #COP5615isgreat
user4 tweeting that #COP5615isgreat
user7 tweeting that #COP5615isgreat
```

[6] Once the user wants to **get tweets belongs to it**, the output information would show like this.



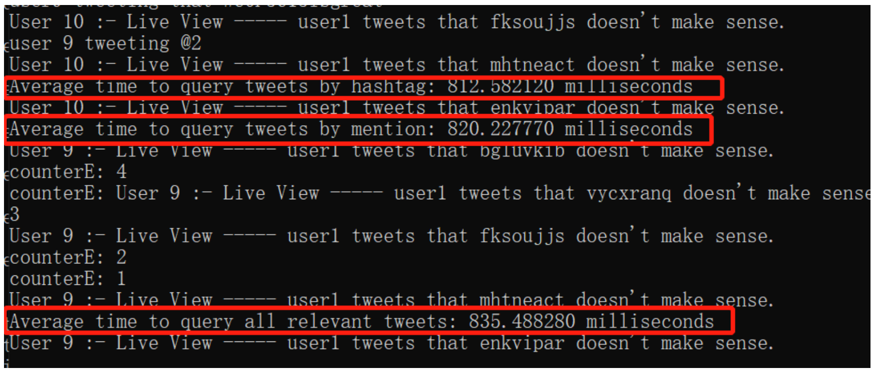
```
User 1 :- All my tweets

counterE: 10
User 10 :- All my tweets
user1 tweets that enkvipar doesn't make sense.
user2 tweeting that #COP5615isgreat
counterE: 9
User 9 :- Tweets Subscribed To
user6 tweeting that #COP5615isgreat
user1 tweeting that #COP5615isgreat
user10 tweeting that #COP5615isgreat
user1 tweets that mhtneact doesn't make sense.
User 5 :- All my tweets
user3 tweeting that #COP5615isgreat
```

[7] Once a user's **state changed from offline to online**, the user would receive tweets from its subscribers.



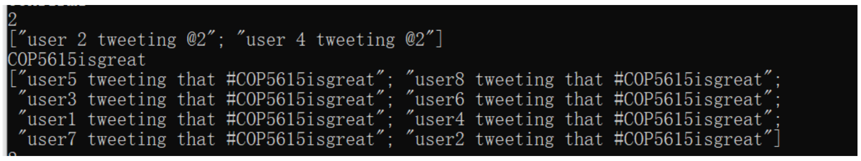
```
User 10 :- Live View ----- user2 tweets that jwxxbdng doesn't make sense.
```

[8] Some needed **time** would also show in the client's output.



```
User 10 :- Live View ----- user1 tweets that fksoujjs doesn't make sense.
user 9 tweeting @2
User 10 :- Live View ----- user1 tweets that mhtneact doesn't make sense.
Average time to query tweets by hashtag: 812.582120 milliseconds
User 10 :- Live View ----- user1 tweets that enkvipar doesn't make sense.
Average time to query tweets by mention: 820.227770 milliseconds
User 9 :- Live View ----- user1 tweets that bgluvklb doesn't make sense.
counterE: 4
counterE: User 9 :- Live View ----- user1 tweets that vycxranq doesn't make sens
3
User 9 :- Live View ----- user1 tweets that fksoujjs doesn't make sense.
counterE: 2
counterE: 1
User 9 :- Live View ----- user1 tweets that mhtneact doesn't make sense.
Average time to query all relevant tweets: 835.488280 milliseconds
User 9 :- Live View ----- user1 tweets that enkvipar doesn't make sense.
```

[9] In the server end. Our program would output some specific tweets list.

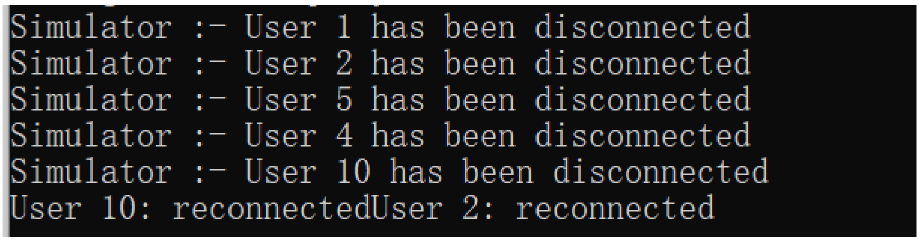


```
2
["user 2 tweeting @2"; "user 4 tweeting @2"]
COP5615isgreat
["user5 tweeting that #COP5615isgreat"; "user8 tweeting that #COP5615isgreat";
 "user3 tweeting that #COP5615isgreat"; "user6 tweeting that #COP5615isgreat";
 "user1 tweeting that #COP5615isgreat"; "user4 tweeting that #COP5615isgreat";
 "user7 tweeting that #COP5615isgreat"; "user2 tweeting that #COP5615isgreat"]
```

Above shows that once the user wants to get all the tweets which **mentioned 2**, it would get a tweet lists contains all the tweets which mentioned 2.

Also, if a user wants to get all the tweets which has a **hashtag** of "cop5615is great", it would get a tweet lists contains all the tweets with this hashtag.

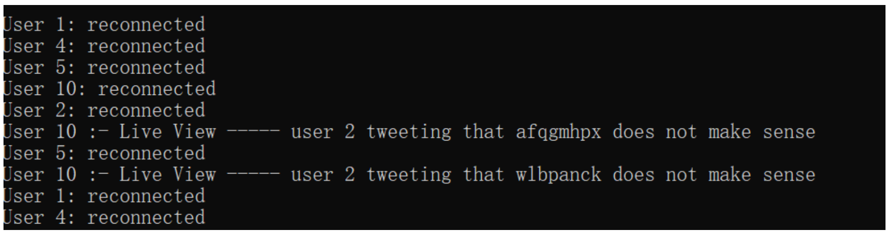
[10] Once we input 10 5 50 which means 50% users would **disconnect** randomly, and the output shows below.



```
Simulator :- User 1 has been disconnected
Simulator :- User 2 has been disconnected
Simulator :- User 5 has been disconnected
Simulator :- User 4 has been disconnected
Simulator :- User 10 has been disconnected
User 10: reconnectedUser 2: reconnected
```

Once the specific user is disconnected, then it would not receive tweets which mention it.

[11] After those users **reconnected**, output would show like below.

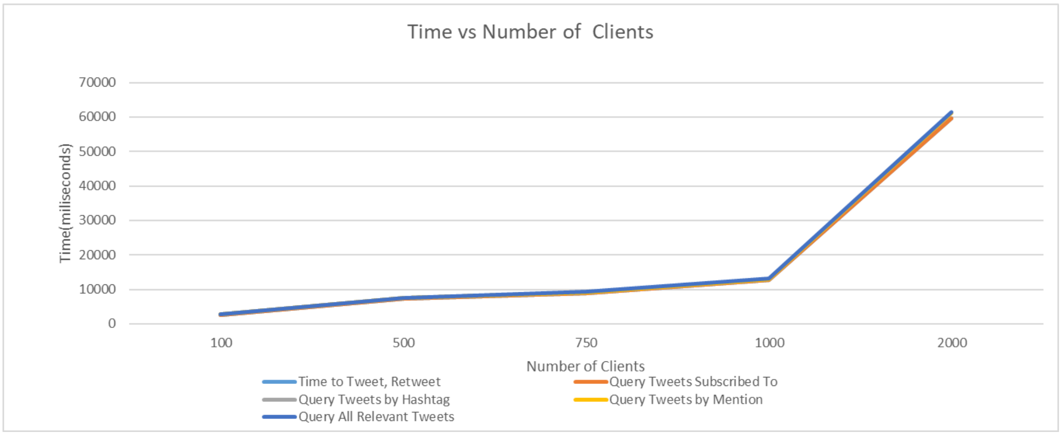


```
User 1: reconnected
User 4: reconnected
User 5: reconnected
User 10: reconnected
User 2: reconnected
User 10 :- Live View ----- user 2 tweeting that afqgmhpx does not make sense
User 5: reconnected
User 10 :- Live View ----- user 2 tweeting that wlbpanck does not make sense
User 1: reconnected
User 4: reconnected
```

The values and consolidated graphs for results we obtained have been disclosed below:

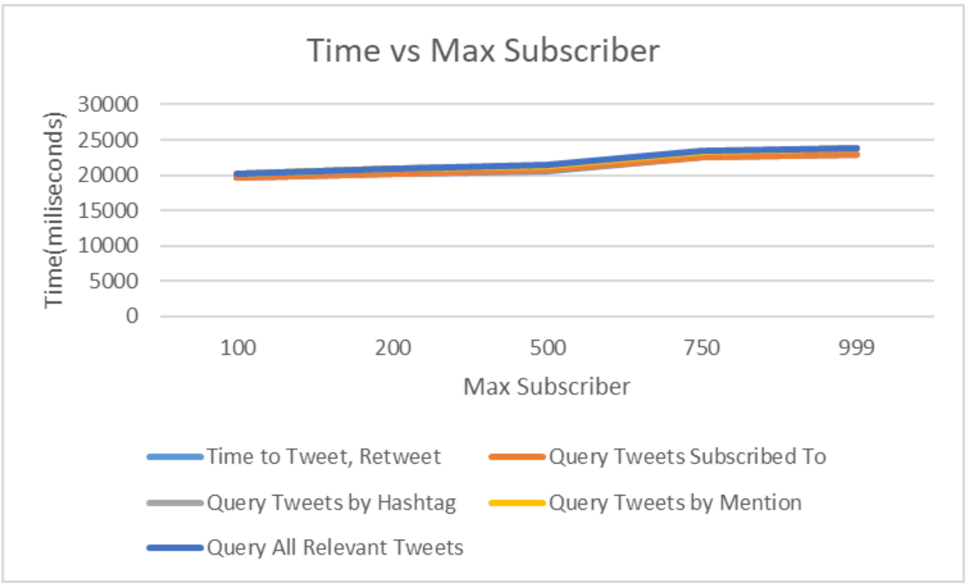
Performance when Max Subscribers(and max tweets/account) = Number of Clients (0% disconnectClients)

Number of Clients	Max Subscribers	Time to Tweet, Retweet (milliseconds)	Query Tweets Subscribed To (milliseconds)	Query Tweets by Hashtag (milliseconds)	Query Tweets by Mention (milliseconds)	Query All Relevant Tweets (milliseconds)
100	99	2613.59	2644.24	2806.27	2810.36	2876.34
500	499	7332.68	7367.45	7517.59	7518.55	7580.41
750	749	8876.8	8910.77	9156.45	9173.77	9244.30
1000	999	12627.04	12670.43	12970.35	12971.31	13062.18
2000	1999	59549.81	59619.24	61244.78	61245.30	61422.27



Performance for different number of Max Subscribers / Max Tweets per Account (0% disconnectClients) is below

Number of Clients	Max Subscribers	Time to Tweet, Retweet (milliseconds)	Query Tweets Subscribed To (milliseconds)	Query Tweets by Hashtag (milliseconds)	Query Tweets by Mention (milliseconds)	Query All Relevant Tweets (milliseconds)
1000	100	19622.05	19624.29	20154.95	20156.17	20261.87
1000	200	20272.81	20281.06	20856.35	20857.19	20957.95
1000	500	20641.27	20667.75	21345.79	21347.68	21468.54
1000	750	22512.7	22566.72	23262.73	23263.98	23400.83
1000	999	22915.04	22979.71	23762.96	23763.5	23900.08



For our system, a decrease was observed in the maximum number of clients supported by our engine when the number of tweets per account was increased, i.e. by increasing the maxSubscribers parameter.