COP5615- Distributed Operating System Principles

Project 4.1

Developed by –
Shaik, Fayaz - UFID: 6326-9935
Mullapudi, Aseesh - UFID: 9175-1971

Brief Description:

The goal of the project is to implement a Twitter clone which includes the Twitter engine and the Twitter client. Web Sockets will be used in the part 2 of this project to integrate this with a user interface. In this project, we have simulated all the operations (Registration, Tweeting etc) and reported the various performance measures by varying the percentage of active users and by also varying the maximum subscribers that a user can have.

Functionalities implemented are:

- Register User
- Login and Logout User
- Subscribe/Follow a User
- Send Tweets. Tweets can have mentions and hashtags
- Retweeting
- The News Feed of the active users is updated real-time and when the user comes online, they can also see the tweets which they have missed
- Querying all the mentions (when the user is mentioned in a tweet) and hashtags

How to Run the Project:

Program.fs – This file contains the main function (entry point). We need to give input for three parameters from the console in this particular order:

- numUsers – the number of users in the system
- maxSubs – the maximum number of subscribers that the user can have
- liveUsersPercentage – The % of users active in the system at any point of time.

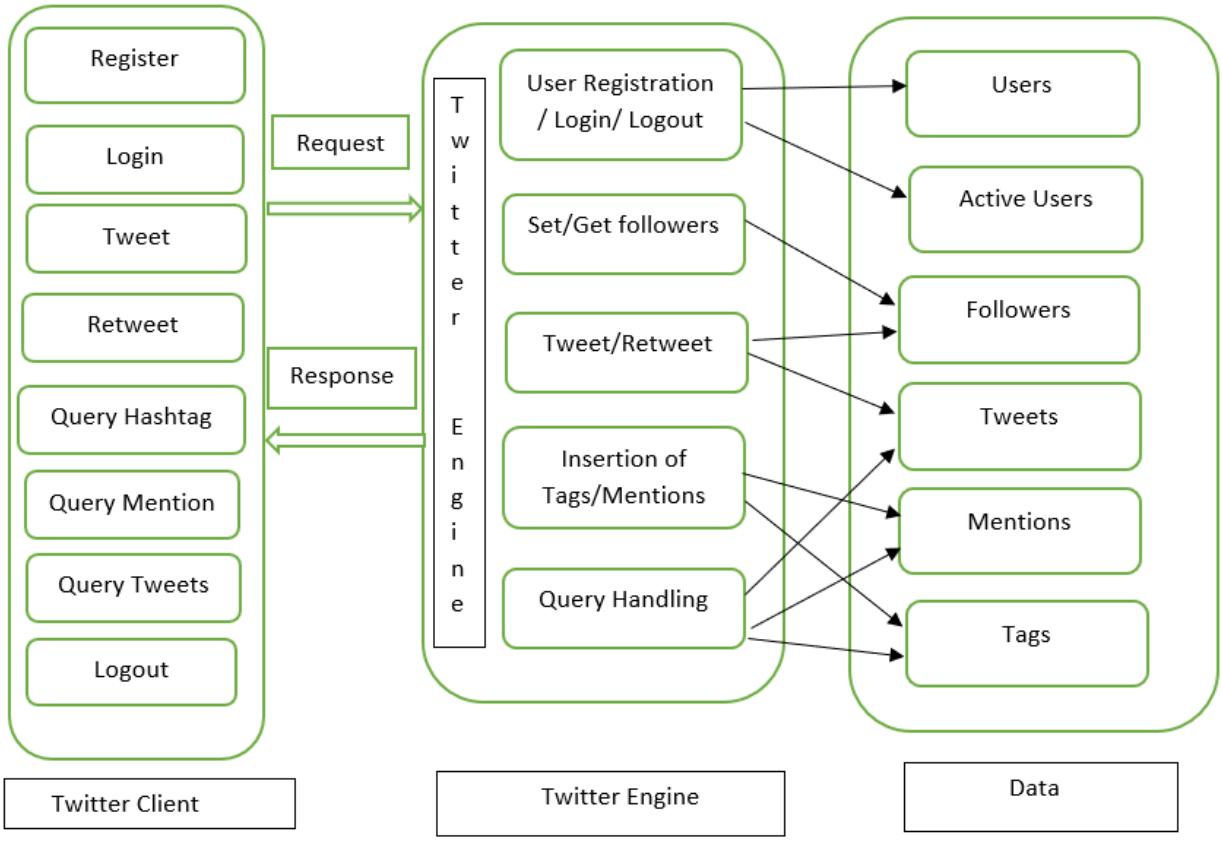
For example – dotnet run < numUsers > < maxSubs > < liveUsersPercentage >

dotnet run 1000 500 90

This would mean that total number of users are 1000, 500 is the maximum Number of followers that a user can have, 90% of all users will be active

The maxSubs should be **strictly less than the numUsers.

Note: the news feed of the user or the tweets done by a user won't be printed to the console as those printfn lines have been commented. You can only see the success operation messages and the execution times in the console.



Architecture

Implementation Details:

1. The ***Program.fs*** file contains the simulation part. The simulation is done by an Actor itself, following operations are done in a sequence:
    - The simulator begins the process by invoking a method in the ***TwitterClient.fs*** The client subsequently spawns the actors, and each user gets registered at the twitterEngine which is in ***TwitterEngine.fs.*** The Simulator receives Acknowledgement from the twitterEngine and then starts the login Process.
    - Login request sent to client then sent to server from client. Receive login ack from server and then user subscription process starts
    - Subscription process starts, receive ack from server
    - Tweeting process starts, receive ack from server
    - ReTweeting process starts, receive ack from server
    - Querying mentions start, receive ack from server
    - End the simulation

2. The ***TwitterClient.fs*** file is where all the client actors are spawned. The Simulator calls the client, and the client passes the type of operation to be done to the server.
Also, each client has an active news Feed and an inactive news Feed. Whenever a user which the client follows tweets, that tweet is pushed into the active news Feed and if our client is inactive or has logged out, that tweet is pushed in the inactive news feed list so that the client can view these tweets when it comes back online. Each client also has a mentions list which on querying collects the tweets in which the user/client is mentioned.

3. The ***TwitterEngine.fs*** is where all the operations take place. Each operation has its own function. All the storage of the data is being done in-memory by using HashMaps. There are. There are maps for –
    - User and user's tweetIds – userId as key and set of tweetIds as value
    - User and user's followers – userId as key and set of followers as value
    - tweetId and tweet – tweetId as key and tweet as value
    - mention and tweetIds – mention as key and set of tweetIds
    - hashtag and tweetIds – hashtag as key and set of tweetIds
    - Set of active Users

    Whenever a user tweets, the tweet is sent to the user's followers if they are active.

    We have also simulated periods of live connection and disconnections for the users
    In the engine, whenever a user tweets, the active users' list is regenerated by taking into account the ***liveUsersPercentage*** which Is passed as an input. For example, 80% of the users should be active at any point of time, when a user tweets the active users list is shuffled and 80% of the list is picked. So, if a certain user is active right now, they might not be active later as the list is shuffled.

    The ReTweet functionality – a user will randomly pick a tweet from their news Feed and tweet it. We have appended "RT" to the tweet being retweeted to differentiate it from the normal tweets. This ReTweet will go through the same process as a normal tweet does. 20% of the total users will retweet one tweet.

    Zipf distribution – As per the requirements of this project, we have simulated a Zipf distribution on the number of subscribers. As mentioned above, we will be taking the max subscribers count(***maxSubs***) as an input parameter which is nothing but the maximum number of followers a client can have in the simulation. We have a used the Mathnet module, which is a dotnet library. We need to provide 2 parameters S and N to this class, and it sample it for each user, we will get a number between 1 and ***maxSubs*** which will be number of followers that the user will have in the system. S is the value of the exponent characterizing the distribution and N is ***maxSubs.***
    Since the users with maximum subscribers should tweet more, the numbers of tweets that a user tweets is also equal to the number of subscribers it has.

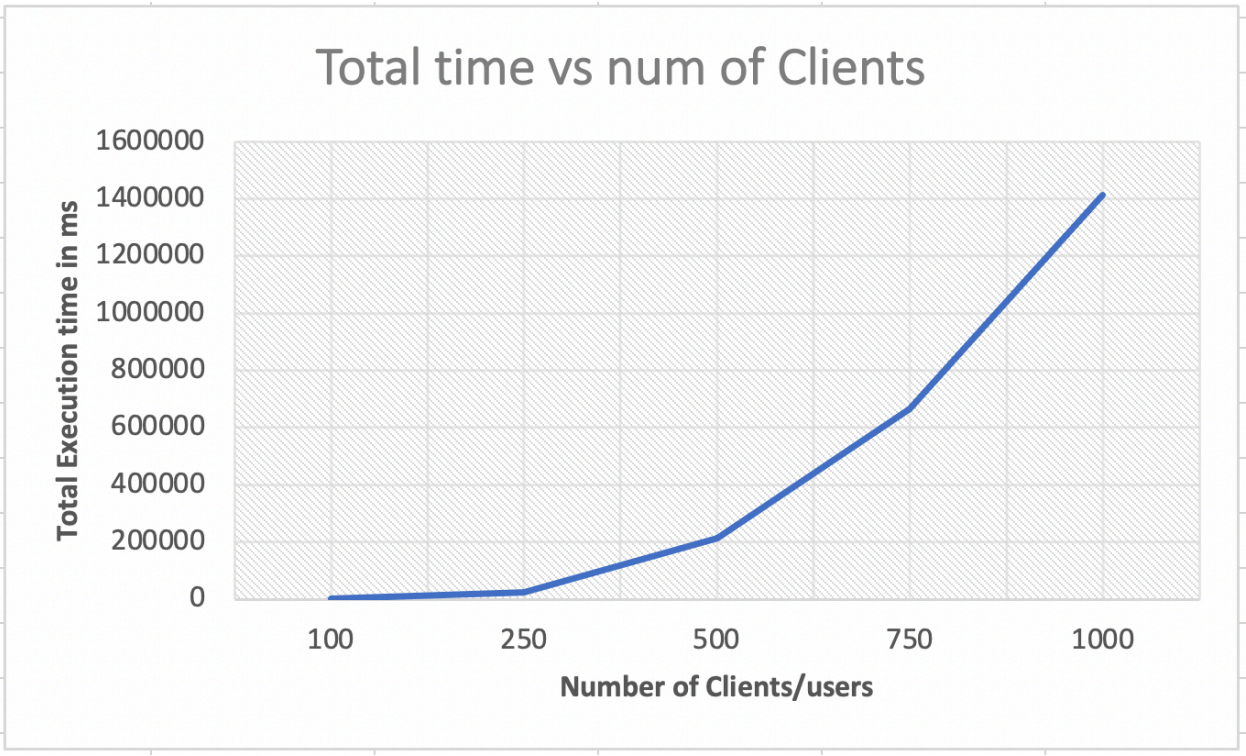
Results:

The total execution time vs the number of clients has been plotted as a graph. The maximum number of users that we could handle is 10000. We could easily handle much more than this if the max subscribers/user is less and of course with more CPU power. The values have been tabulated and the graph is plotted below:

The performance when Max Subscribers per user is equal to num of users – 1.
100% users are active

Number of Clients	Max Subscribers	Time to Tweet(ms)	Time to Subscribe(ms)	Time to Query (all mentions)	Total Execution time (ms)
100	99	1860	48.041	6.26	1976
250	249	23178	256.74	17.63	23613
500	499	212594	882	64.58	214133
750	749	663059	1991.172	147.31	666359.81
1000	999	1406849	3227.514	293.22	1412478

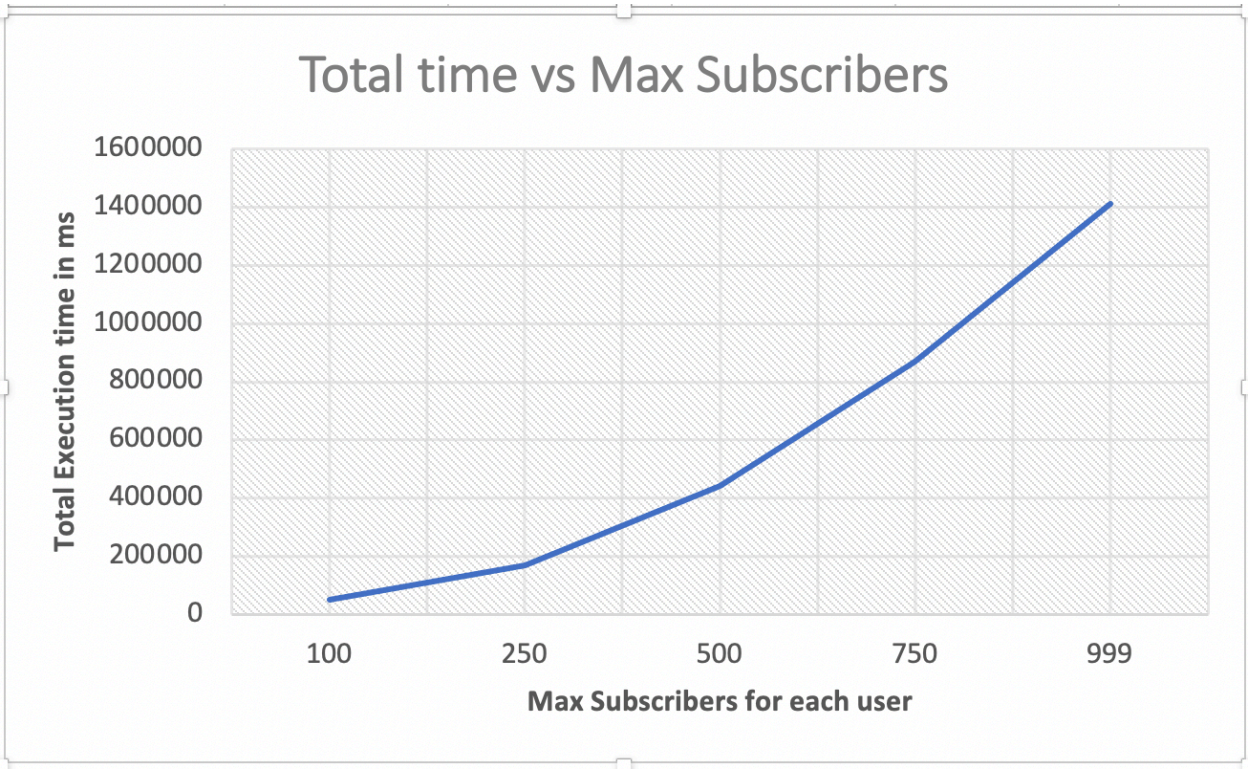
All the times are in milliseconds



As the number of clients or users is increasing, we have observed that the total time is also increasing.

The total time of execution vs the maximum number of subscribers that a user can have.
The number of users has been set to 1000.
100% of the users are active

No. Clients	Max Subscribers	Time to Tweet(ms)	Time to Subscribe(ms)	Time to Query (all mentions)	Total Execution time (ms)
1000	100	49272	595.597	62.419	50729
1000	250	167084	1255.2	194.9	168580
1000	500	439856	1975.98	144.78	443182
1000	750	866053	2444.72	211.71	870355
1000	999	1406849	3227.514	293.22	1412478



We found that as the max subscribers per user increased, the total execution time also increased. But, a greater number of users can be supported if the max subscription per user (num of tweets per user) decreased.