COP5612 – Fall 2020
Project 4 Part 1 - Twitter Clone and a client tester/simulator

Kanika Sharma, UFID: 7119-1343

Instructions to run the program :

Go to directory TwittercIonePart1 by using the command :

```
cd TwittercIonePart1
```

Ensure Client.fsx, Server.fsx, Simulator.fsx and Messages.fsx are included in the compile in .fsproj file

Through the command line, run the program using the commands :

First run the server process using :

```
dotnet fsi --langversion:preview Server.fsx
```

After server has started running, run the simulation file using :

```
dotnet fsi --langversion:preview Simulator.fsx numClients maxfollowerCount maxtotaltweets
```

(Note : Provide values for all 3 parameters :

1. numClients - number of users the simulation will have
2. maxfollowerCount - maximum number of users a user can follow
3. maxtotaltweets - maximum number of tweets a user can tweet)

Project Description :

The project is an implementation of a Twitter clone and a client simulator. The twitter engineer provides functionalities like registering user accounts, sending tweets (which can contain tweets with hashtag and user mentions), subscribing to user's tweets, retweeting, querying "subscribed to" tweets, hashtag tweets and usermention tweets.

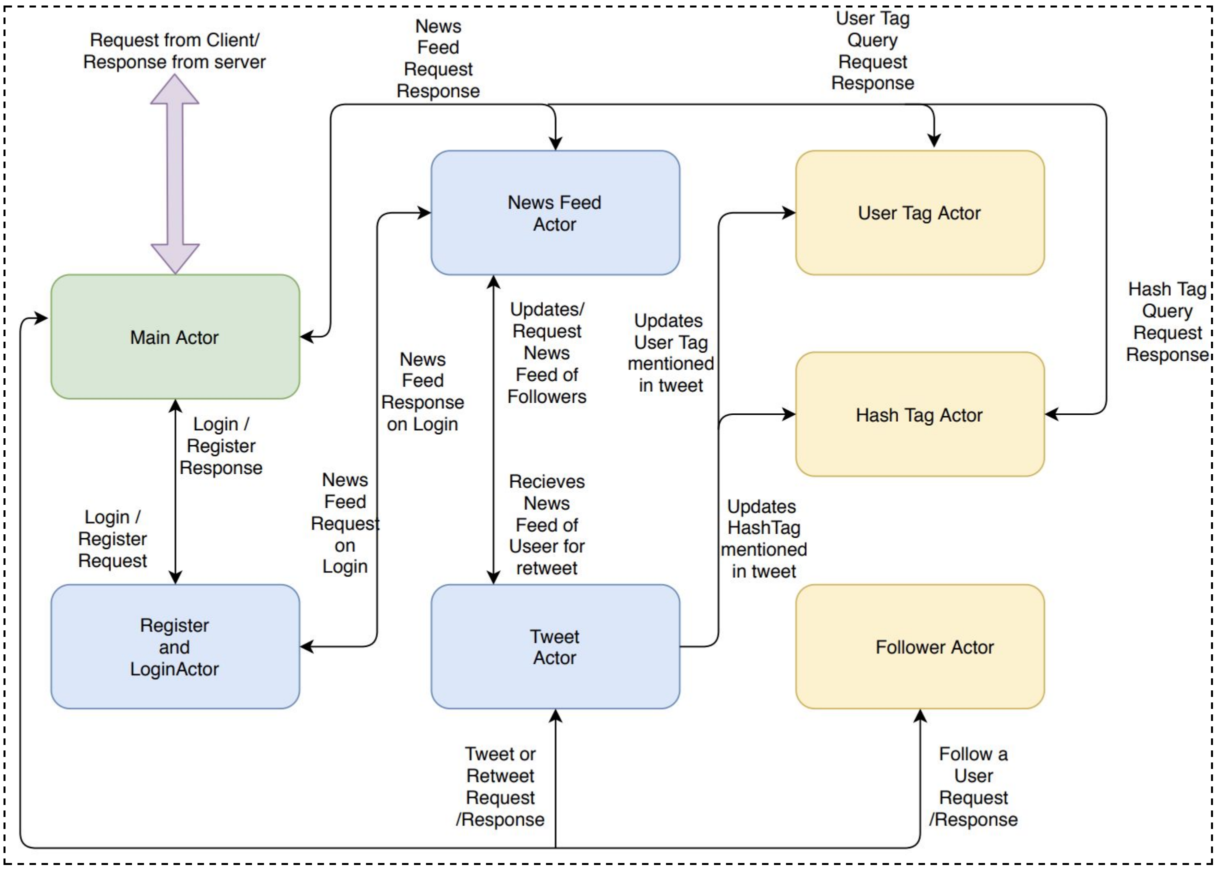
Twitter Clone Architecture :

The project has been implemented like microservices architecture. The project has two parts - the Client part and the Twitter engine or Server part. The actors act like a service and provide the required functionality. These two are implemented as separate processes inside the project.

Twitter Engine or Server :

The server handles the requests obtained from the users. The server starts running at a port and the main actor in the server receives requests from the user. The request is received from the client in JSON format. The main actor deserializes the received string and then takes the decision of sending it to the right actor/service. There are separate actors for each client request such as user registration, user login, subscribing, sending tweets, querying, and logout. Each of these functions uses a hashmap to store the information. Whenever the client requests some information the data is fetched from the hashmap, serialized, and sent from server to client.

The diagram below describes the low-level server architecture with all the actors/services.


Request from Client/ Response from server
News Feed Request Response
User Tag Query Request Response
News Feed Actor
User Tag Actor
Main Actor
Hash Tag Query Request Response
Updates/ Request News Feed of Followers
Updates User Tag mentioned in tweet
News Feed Response on Login
Hash Tag Actor
Login / Register Response
Login / Register Request
News Feed Request on Login
Recieves News Feed of Useer for retweet
Updates HashTag mentioned in tweet
Register and LoginActor
Tweet Actor
Follower Actor
Tweet or Retweet Request /Response
Follow a User Request /Response

Below is a table showing the actors we spawn for each of the functions and the hashmaps we have used :

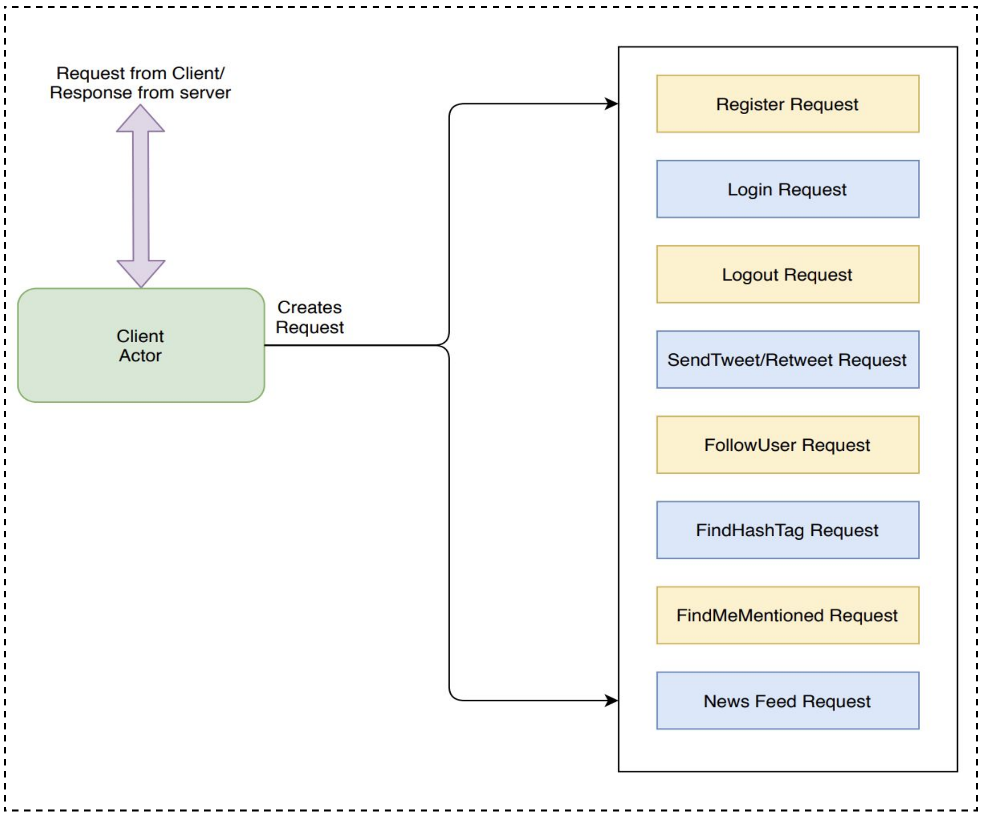
Actors	Functionality	Data Structure(Dictionary)
registerLoginActor	Performs registration of new	regDict

	user or logs him in if already present and if password matches.	(userId,Password)
followerActor	Serves the request of a user for following another user	followerDict (UserId,Set<userIds>)
sendTweetActor	1) Performs the task of tweeting for a user. Stores the new tweet in the dictionary and generates tweet Ids. 2) Passes the tweet to hashTag and UserTag actors to search for mentioned @users or #HashTags. 3) Adds to the news feed Table of followers.	tweetDict (tweetId, Tweet)
hashTagActor	Parses Tweet to search for any HashTags and stores the hashTag and the tweetIds which mention that hashTag.	hashTagDict (HashTag, Set<TweetIds>)
userTagActor	Parses Tweet to search for any User who has been tagged and stores all the tweetId where the user has been mentioned. Responds to user tagged Query by searching	userTagDict (userId,Set<TweetIds>)
newsFeedActor	Save the newsFeed of each user based on who all the user follows. Displays newsFeed to each user on login. Responds to newsFeed request by providing tweets of the people user follows.	newsFeedDict (userId,Set<tweetIds>)

Twitter Client :

The client sends a request at the same port where the server is listening. The client performs the functionalities of creating a new user, logging in, tweeting, retweeting, requesting his news feed, following a user, query for HashTag, or query for his mention. The request is serialized and sent as JSON to the server. The client also receives a response from the server.

The diagram below describes low-level architecture of client functionalities.



For each user a client Actor is spawned. The client creates the request and sends to server. The client also receives a response from the server.

Request	Functionality	Response
RegisterRequest	This sends the username and password to the server.	The server saves the user and its password and returns a RegisterRequest Response.
Login	The user enters his userId and password.	If the password matches then user is logged in and newsfeed of user is sent in response to be displayed els user can't login.
Logout	This request logs out users from the active list.	The client receives a response of logout.
SendTweet	This request is done by the client when the tweet is to be published. The request contains userId and tweet string	The client receives a response of sendtweet to confirm the task has been done.

FollowUser	This request is sent by user when user1 wants to follow some other user like user2.	The client receives a response in affirmation to this.
FindHashTag	This request is to query and find all tweets of a particular #Tag like #DistributedSystems	The client receives all the tweets of the #Tag as response.
FindMeMentioned	This request is to query and find all tweets the user himself has been mentioned.	The client receives all the tweets of the where he has been mentioned by followers as a response.
Retweet	This request will query for the tweets of the people the user follows.	The client receives the list of all the tweets of the people he follows. Then the user selects one tweet randomly and performs a SendTweet request for a new tweet.
NewsFeed	This request is made when a user explicitly requests for news feed.	The user receives all the tweets of the people he or she follows.

Simulation :

As part of the project, we have created a client tester/simulator that takes in 3 inputs:

4. numClients - number of users the simulation will have
5. maxfollowerCount - maximum number of users a user can follow
6. maxtotaltweets - maximum number of tweets a user can tweet

For each user, the number of users it should follow is calculated using the parameters numClients and maxfollowerCount. Not every user follows every other user. Users with higher rank (user ids in the initial range like 1,2,3, etc) are followed by more number of people than users with lower rank. And the users with higher rank follow very few users or mostly none. This way we achieved a zipf distribution on the number of subscribers.

For each user, the number of tweets it should do is calculated using the parameters numClients and maxtotaltweets. Not every user does the same number of tweets. Users with higher rank tweet more than users with lower rank. Users with lower rank can see tweets shared by their followers. Any user can retweet a tweet posted by its followers. As part of the simulation, we have made a few tweets as retweets. Also some tweets contain hashtags and some tweets contain user mentions. Tweets are randomly generated and hashtags are fetched randomly from a pool of hashtags we have created for simulation.

We have done 3 types of queries as per the requirement - Query tweets subscribed to, hashtag tweets and usermention tweets. We have calculated the performance for each of these functionalities while all users are connected (logged in). When the user is logged in, these tweets are shown live as well.

After performance is calculated, to simulate periods of live connection and disconnection, we disconnect (logout) 20% users and let the disconnected 20% users login again and post tweets and disconnect another 20% users randomly and connect them again each with a sleep time of 2000ms. This will continue to run in a recursive loop.

Results :

We have measured various aspects of our simulator and calculated the performance. We were able to simulate a maximum of 100,000 users.

Below is a table showing the runtime for each functionality for different number of users (Time is listed in milliseconds) :

Performance with respect to **registration and login**(milliseconds) :

Number of users	Average time taken to register users	Average time taken to login users
5	399.6	397
10	382	380
100	319.42	317.05
1000	441.296	438.681
10,000	2186.3834	2184.173600
100,000	20588.245650	20585.615170

Performance with respect to **subscribing**(milliseconds) :

Number of users	Maximum subscribers per user	Total number subscribing to (Based on zipf distribution)	Average time taken to do subscribing
5	4	4	38
10	9	14	57.071429
100	99	374	46.810160
1000	999	6054	380.674595
10,000	1100	6781	353.340658
100,000	1100	6781	330.900015

Performance with respect to **tweeting**(milliseconds) :

Number of users	Maximum number of tweets per user	Total number of tweets	Average time taken to tweet
5	4	8	74
10	9	23	88.217391
100	99	473	62.243129
1000	999	7053	566.102935
10,000	1100	7881	652.548534
100,000	1100	7881	646.951021

Performance with respect to **Querying**(milliseconds) :

Number of users	Number of “subscribed to” queries	Average time to do query “subscribed to”
5	5	2064.6
10	10	2029.5
100	100	2029.86
1000	100	2053.82
10,000	10	2049.00
100,000	10	2061.00

Number of users	Number of hashtag tweets	Number of hashtag queries	Average time to do “hashtag” query
5	5	5	80
10	10	10	43
100	100	100	21.96
1000	1000	1000	453.553
10,000	10,000	100	1878.96
100,000	100	100	17.7

Number of users	Number of user mention tweets	Number of user mention queries	Average time to do "user mention" query
5	5	5	75
10	10	10	33.7
100	100	100	26.94
1000	1000	1000	57.103
10,000	10,000	100	1160.57
100,000	100	100	15.46

Graph showing Zipf distribution (user rank vs number of followers) :

Fig 1 :Graph for a simulation of 100 users and Maximum number of tweets per user = 99 :

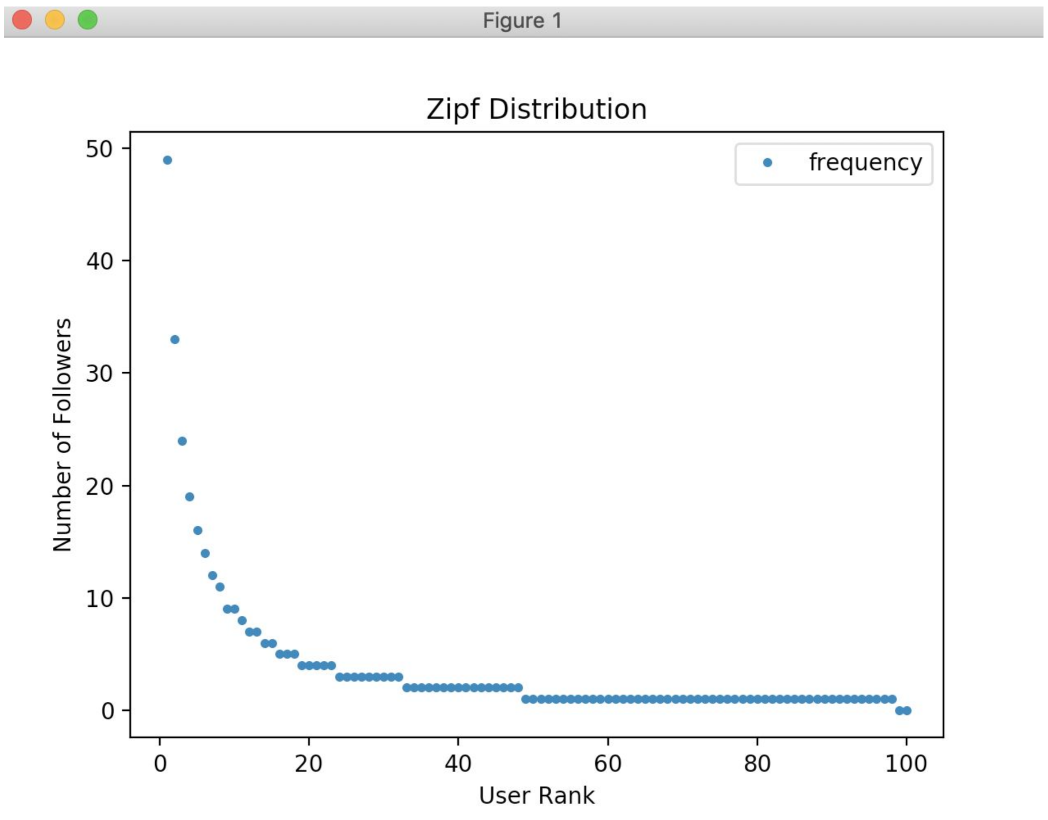


Fig2 : Graph for a simulation of 1000 users and Maximum number of tweets per user = 999 :

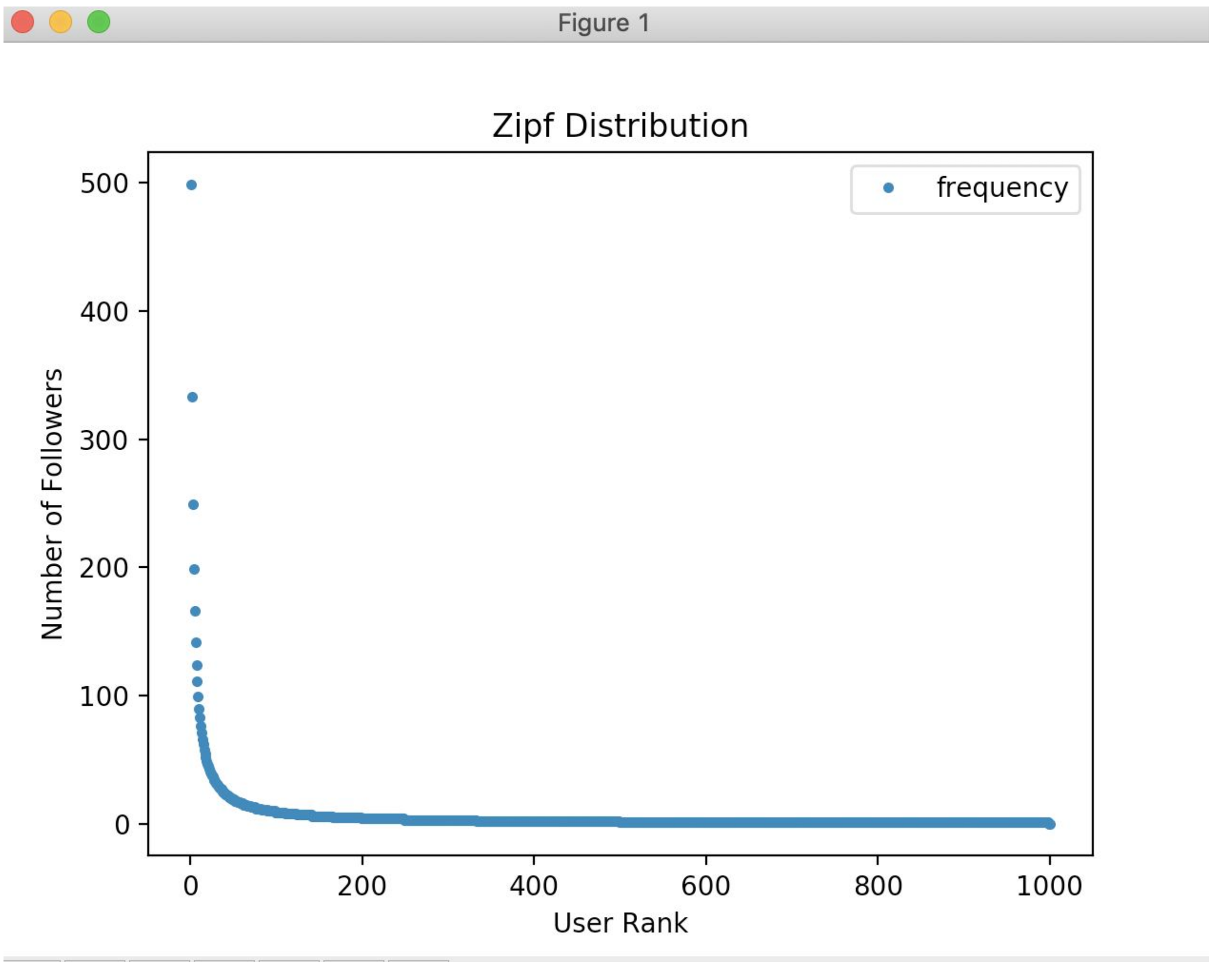
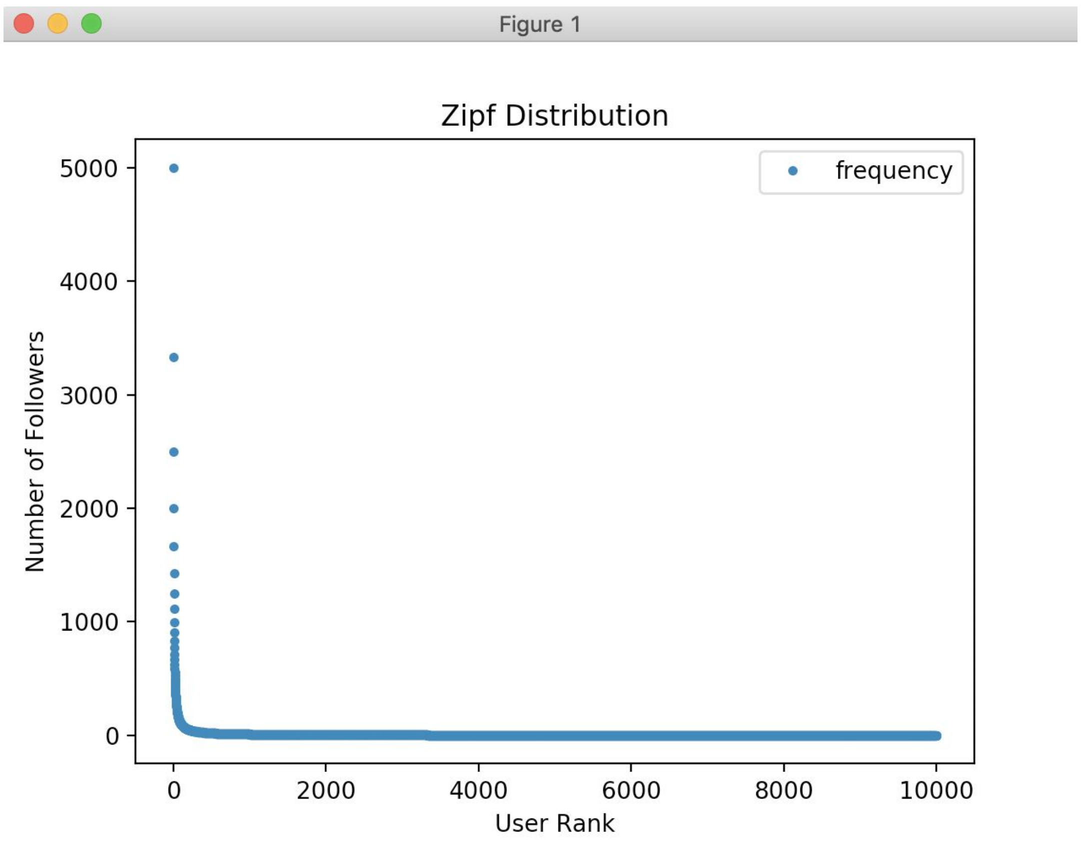


Fig2 : Graph for a simulation of 10000 users and Maximum number of tweets per user = 9999 :



From the graph, we can conclude that our simulation is following Zipf distribution. Users in the range starting from 1 have higher rank and have more followers. As it reaches users with lower rank like 100, 1000 and 1000, the number of followers gradually decreases to 0.