Twitter Clone Project Report

Kratika Singhal 69535971
Shrishail Zalake 07466343

Implementation Scheme

The aim of this project is to develop a twitter clone and demonstrate it using a client simulator. We are using remote client functionality to achieve this. **The server and client are executed as different processes.** The structure is divided into three parts

- **The Server:** server mimics the functionality of a global server which servers all the requests made by the client. The request includes functionalities like register, tweet etc.
- **The Client:** The client here is a remote client which hosts more than 1000 users. These users are log in and log out randomly. Once they are logged in, they make requests to the server.
- **The simulator:** Simulator helps in demonstrating the twitter functionality and triggers different user operations.

The simulator demonstrates the following functionalities – login, log out, tweet, retweet, Query hashtag, query subscribers, query mentions (other users). The simulator selects these operations randomly.

Functionalities included

Register user: every user is an actor in our implementation. All the users are hosted on the client machine. The users are initialized and registered with the server.
Output on server screen when users are registered:

```
Registered user number 0
Registered user number 1
Registered user number 2
Registered user number 3
Registered user number 4
Registered user number 5
```

Output on client machine when users are registered and subscribed:

```
Initializing the users and its subscribers according to zipf distribution
```

Subscribers: Subscriber functionality is created based on zipf distribution. This is done soon after the register process.

Login/Logout: The registered users can login and lout at random time intervals. Once the user is logged in or logged out, a message is displayed on the server screen.
Output on server screen when users logged in and logged out respectively.

```
User 119 logged in
```

```
User 671 logged out
User 27 logged out
```

Tweet: The user can tweet any random tweet. The tweet consists of three components – tweet text, hashtag and mention. The tweet is generated randomly. When a user tweets, it is broadcasted to all its subscribers.

Output on server screen when a tweet is done by any users

```
Tweet done by user number 96
Tweet done by user number 124
```

Output on client screen and any tweet is done. That tweet is sent as a live feed to all the subscribers of user who has tweeted.

```
live feed from user 96 to 344,742, received
live feed from user 124 to 622,751, received
```

Query a hashtag: The user can query a hashtag. In response, all the tweets that contain the queried hashtag are given as a response to the user.

Output on server screen when a user queries a hashtag

```
Hashtag queries by user 355 #evuwria
```

Output on client screen for the same

```
Query response received for #evuwria query
```

Query subscribers: The user query all the tweets from all users whom he or she has subscribed to. In response, all the tweets from those users will be given to the querying user.

Output on server screen when subscribers are queried by any user

```
subscribers queried by user 943
```

Output on client screen when response for subscribers is received at the client side

```
Query response received for subscribers query
Query response received for subscribers query
```

Query mentions: The user can query a hashtag. In response, all the tweets that contain the mentioned user is given as a response to the querying user.

Output on server screen when a user mentions another user in its query

```
user 186 mentioned the following actor in its query 725
user 900 mentioned the following actor in its query 672
user 472 mentioned the following actor in its query 17
```

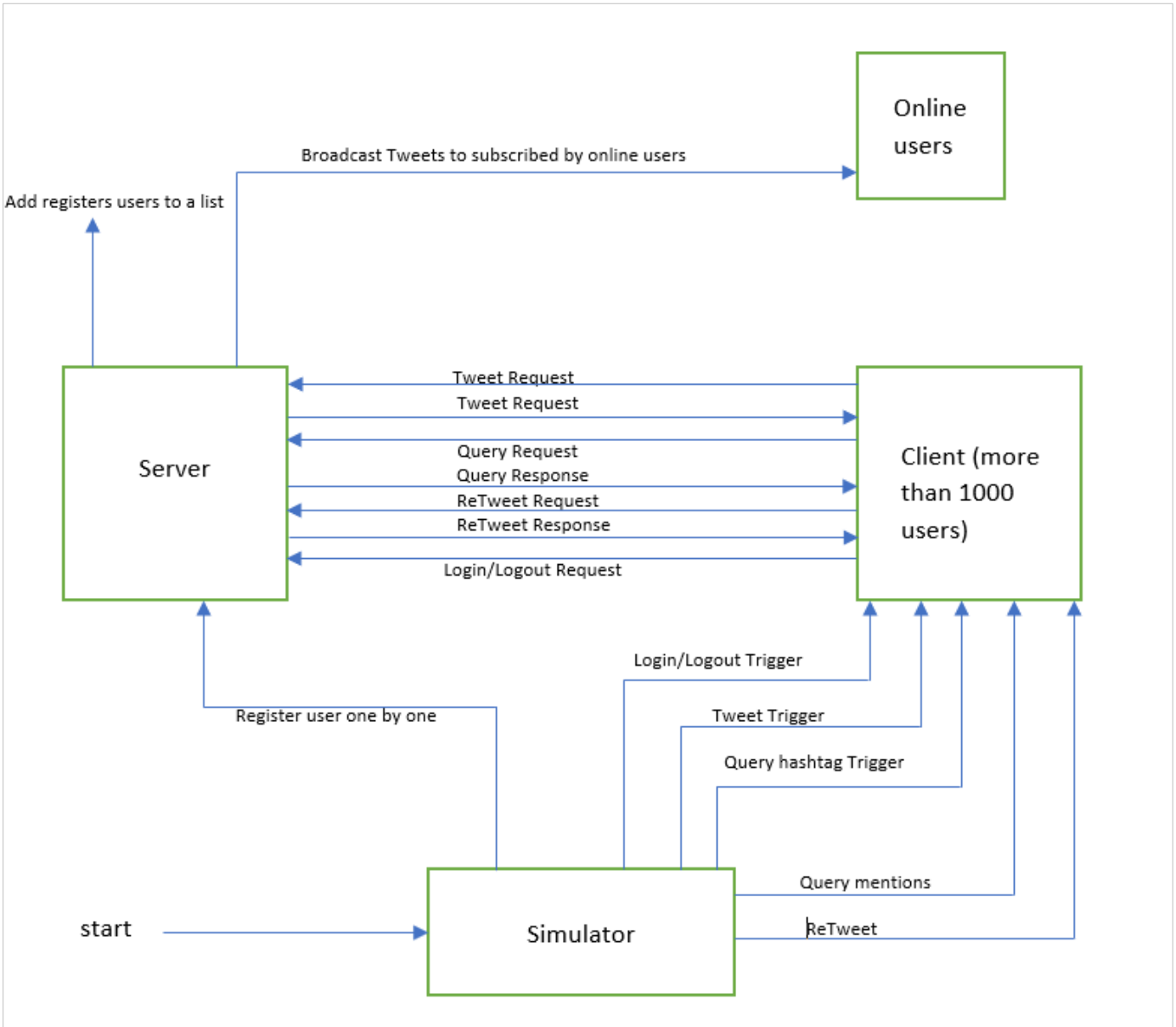
Output on client screen when the response is received for the same.

```
Query response received for mentioned user 725 query
Query response received for mentioned user 672 query
Query response received for mentioned user 17 query
```

ReTweet: A user can retweet another user's tweet. A message will be printed on client screen as shown once the operation is done.
Output on client screen when any user retweets.

```
user 338 retweeted tweet by user 69
```

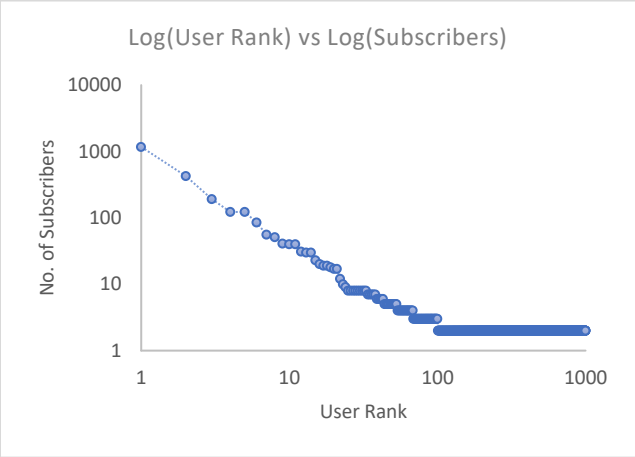
Basic architecture

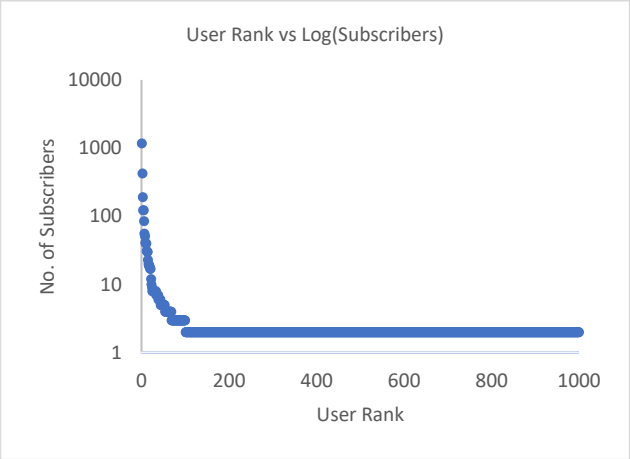


Zipf implementation

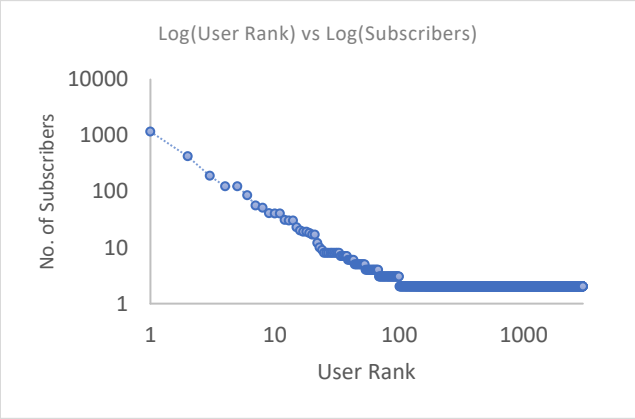
According to Zipf distribution "the frequency of any word is inversely proportional to its rank in the frequency table. Thus, the most frequent word will occur approximately twice as often as the second most frequent word, three times as often as the third most frequent word"
In our case, the user with highest rank will have highest number of subscribers. Also, the user with a greater number of subscribers will tweet more often as compared to the others.

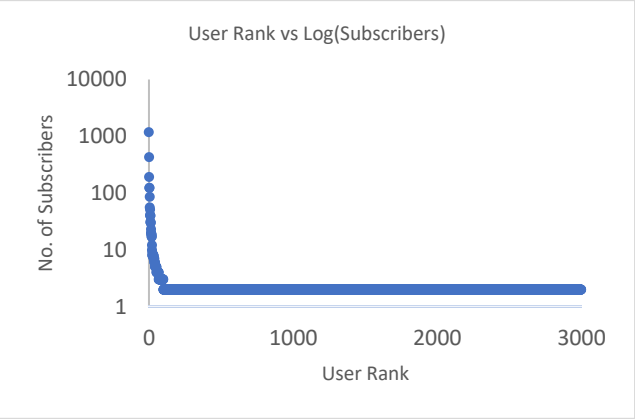
Zipf implementation graph for 1000 users:





Zipf implementation graph for 3000 users:





Other Performance Statistics

For 100 users, 100 operations:

```
elapsed 93 ms
Total Tweets processed 45
Total Queries processed 31
Time per operation 1.075269
```

For 1000 users and 1000 operations:

```
elapsed 713 ms
Total Tweets processed 447
Total Queries processed 339
Time per operation 1.402525
```

For 3000 users and 3000 operations:

```
elapsed 6232 ms
Total Tweets processed 1332
Total Queries processed 977
Time per operation 0.481386
```

Instruction to Run

To run this program, we first start the server in a terminal as shown below.

dotnet fsi –langversion:preview server.fsx

```
C:\Users\skrat\Project4.1>dotnet fsi --langversion:preview server.fsx
Real: 00:00:00.000, CPU: 00:00:00.000, GC gen0: 0, gen1: 0, gen2: 0
Real: 00:00:00.000, CPU: 00:00:00.000, GC gen0: 0, gen1: 0, gen2: 0
[INFO][12/4/2020 1:11:18 PM][Thread 0001][remoting (akka://RemoteFSharp)] Starting remoting
[INFO][12/4/2020 1:11:18 PM][Thread 0001][remoting (akka://RemoteFSharp)] Remoting started; listening on addresses : [akka.tcp://RemoteFSharp@127.0.0.1:9004]
[INFO][12/4/2020 1:11:18 PM][Thread 0001][remoting (akka://RemoteFSharp)] Remoting now listens on addresses: [akka.tcp://RemoteFSharp@127.0.0.1:9004]
[akka://RemoteFSharp/user/Server#831737861]
```

Once the server is started, we start the client on another terminal as shown below.
We give number of operations and number of users as input

dotnet fsi –langversion:preview <number of operations> <number of users>

```
C:\Users\skrat\Project4.1>dotnet fsi --langversion:preview client.fsx 100 100
^C
```

Please note that the client and server are run on separate terminals. Also, please install the following nuget packages for the remote client and zipf distribution to work.

- Akka.Serialization.Hyperion
- MathNet.Numerics
- Akka.Remote

```
#r "nuget: Akka.Serialization.Hyperion"
#r "nuget: MathNet.Numerics"
```

Output

For the scope of report, we have restricted the number of operations and number of users to 20 each. However, in the project both number of users and number of operations is a user given input. User specifies the number of operations and simulations performs the operation of tweet, query, retweet etc. that many numbers of times randomly.

Output on the client side when number of users and operations are both 20

```
Initializing the users and its subscribers according to zipf distribution
start simulation
user number 17 quereing hash tag #q
user number 5 quereing hash tag #rs
Query response received for mentioned user 1 query
Query response received for subscribers query
Query response received for subscribers query
live feed from user 14 to 6,3, received
Query response received for #q query
Query response received for subscribers query
live feed from user 8 to 19,14, received
live feed from user 10 to 13,14, received
live feed from user 19 to 15,14, received
live feed from user 10 to 13,14, received
Query response received for subscribers query
Query response received for mentioned user 17 query
Query response received for mentioned user 2 query
Query response received for #rs query
live feed from user 2 to 7,14,6,1,4, received
live feed from user 10 to 13,14, received
time taken to complete 20 operations: 157 ms
Please press any key to end the simulation
```

Output on the server side. Users are registered first and then the operations are performed accordingly.

```
Registered user number 0
Registered user number 1
Registered user number 2
Registered user number 3
Registered user number 4
Registered user number 5
Registered user number 6
Registered user number 7
Registered user number 8
Registered user number 9
Registered user number 10
Registered user number 11
Registered user number 12
Registered user number 13
Registered user number 14
Registered user number 15
Registered user number 16
Registered user number 17
Registered user number 18
Registered user number 19
user 5 mentioned the following actor in its query 1
subscribers queried by user 13
subscribers queried by user 14
Tweet done by user number 14
Hashtag queries by user 17 #q
subscribers queried by user 5
Tweet done by user number 8
Tweet done by user number 10
Tweet done by user number 19
Tweet done by user number 10
subscribers queried by user 14
User 19 logged out
user 1 mentioned the following actor in its query 17
user 0 mentioned the following actor in its query 2
Hashtag queries by user 5 #rs
Tweet done by user number 2
Tweet done by user number 10
elapsed 79 ms
Total Tweets processed 8
Total Queries processed 9
Time per operation 0.253165
```