# COP5615 – Distributed Operating Systems Principles

## Project 4 Part 1 Report

**Submitted By:**

Bhuvana Venkatesh, UFID: 1687-2832

Yashaswini Kondakindi, UFID: 7061-1190

### Objective of the Project

Our Twitter engine consists of several features such as client registration, tweets, following or subscribing, retweets, hashtag searches, and mentions of other Twitter users. As a means of pushing live feeds to online users as well as notifications, actors are used.

The process of sending and receiving tweets, as well as the process of dispersing them, should be separated from each other. The use of a single engine and multiple independent client processes enables the simulation of thousands of clients at the same time using a single engine.

In Erlang, we use a List<Dictionary<Dictionary>>> b structure as a data storage mechanism to store our data. Maps, Sets, and Lists are Erlang's data structures which are used for storing and manipulating information.

### Steps To Run The System dependencies

- We will need to unzip the project4.zip folder to get access to the engine.erl and client.erl files.
- A shell environment for Erlang should be used for the execution of the project.
- It makes use of multiple cores in the CPU. Although this is not necessary, multicore systems can significantly speed up the processing speed of the program.

### Compilation of the project

***c(client).***

***c(engine).***

***engine:server_start().***

***Client:simulate(Numberofusers).***

### Running of the project

Upon compilation, run the project by calling the module's main function.
This function takes two arguments, namely the number of nodes and the number of requests.
Use client:simulate(#numOfNodes) in order to run the project.
As a result of the program, the output will be printed on the console and appended to the stats.txt file for later analysis.

## Simulator Design

- For testing, create a tester/simulator
- As many users as possible should be simulated.
- The Zipf distribution should be based on subscriber numbers.
- A live period and a disconnection period should be provided to users.
- Increase the number of tweets sent by accounts with a significant number of subscribers. The messages can be retweeted in some cases.

## Functions Description

There are two files that make up the given implementation:

- *engine.erl* : Defining an interface for interacting with a system implemented in the form of a twitter-like system for sharing tweets. It is responsible for defining the semantics of the language and it should remain the same.
- *client.erl :* Based on a single data actor, this is a simplified example implementation demonstrating the use of an Erlang process to provide a basic service.

## Twitter Engine:

- We use the Twitter Engine to provide users with the ability to register, subscribe, tweet, retweet, query, hashtag, mention other users, log in, and log out of Twitter.
- Different services are handled by different actors, i.e., for every action, there is a unique actor that handles the corresponding request.
- Server requests are handled by the request handler. Requests are directed to the appropriate actors based on incoming requests.
- The server stores a user's subscriber list, tweets, retweets, mentions, and hashtags.

## Twitter Simulator:

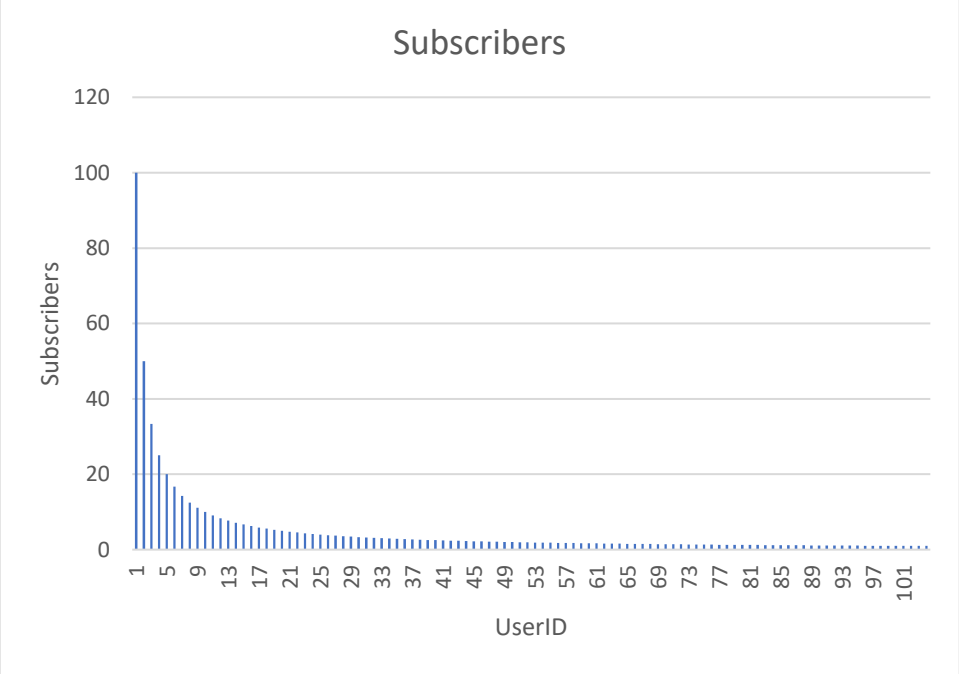
- The Twitter client first spawns a given number of users.
- Users cannot perform any action without first registering.
- A user may then tweet, retweet, subscribe to another user, mention another user, logoff, login, query for mentioned tweets, query for tweets with specific hashtags, and search for subscribed tweets.
- For tweeting and retweeting we have pre-loaded some of the hashtags and tweet messages on the client side.
- Multiple clients can be used to simulate multiple users.
- **Register:** By using the register function call, a user can join the network by contacting the backend.
- **Send Tweet**: This function call provides the code for a tweet to be sent by the user. Users who are subscribed to this user will automatically receive updates regarding the tweet.
- **Subscription**: This function ensures that the client can subscribe to another user. As a result, any tweets made by the subscribed user will be displayed on the terminal
- **Re-Tweet:** This function allows you to retweet the tweet from another user. A MessageID will appear in the bash window for every tweet the user receives.
- **Search**: This can be used to obtain a list of total tweets. Mentions and hashtags are the types of queries. term is the term the user wishes to search for.
- **Terminate**: The terminate () function call terminates the session and disconnects the user from the network.

**Implementation of the project And Observed Output**

- Basic requirement is Completely working
- The server handles the registration process such as logins, logouts, adding followers, sending tweets to other clients, and retweeting.
- Client: An application client sends all the requests to the server and receives a response from the server. It is a simulator module that creates clients, registers them, subscribes them, as well as tweets them.
- Server: Server handles requests from clients and performs various operations distributing work to workers.
- It takes the hashtags and extracts them from the tweets, then retrieves the tweets that mention the hashtags.
- The connection to multiple clients to the server has been successful.
- In terms of number of clients, the largest number tried was 20000.

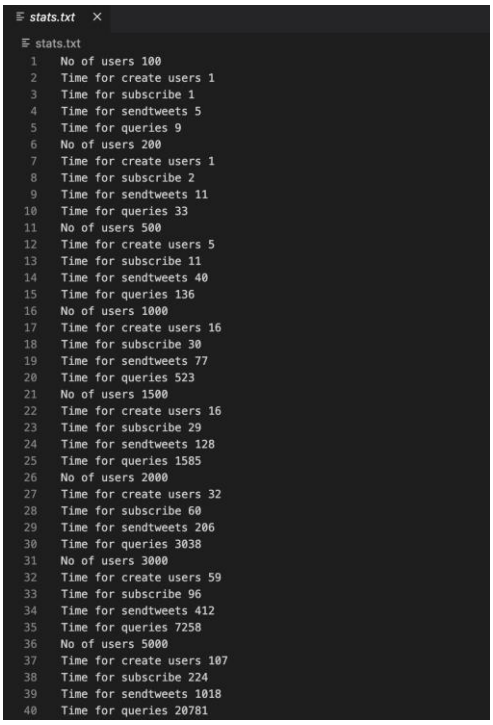
**Simulation:**

The client simulator(serverid, noofusers) simulates users, subscriptions, tweets/retweets, queries, and disconnects. Subscribers to users are based on the formula (TotalNoOfUsers) * (1 / Id). If there are 100 users, user id 1 has 100 subscribers, user id 2 has 50, etc. Tweets are proportional to subscribers. Then, hashtags and user mentions double user queries. Under observations, the times for each step are listed.



## Performance results –

## Onservations and Statistics:



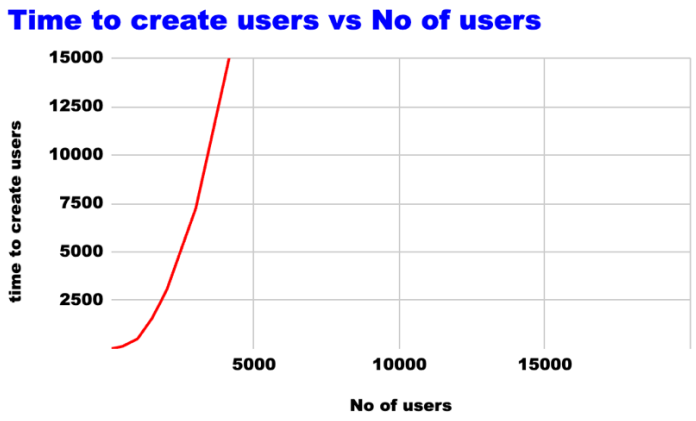
```
stats.txt
No of users 100
Time for create users 1
Time for subscribe 1
Time for sendtweets 5
Time for queries 9
No of users 200
Time for create users 1
Time for subscribe 2
Time for sendtweets 11
Time for queries 33
No of users 500
Time for create users 5
Time for subscribe 11
Time for sendtweets 40
Time for queries 136
No of users 1000
Time for create users 16
Time for subscribe 30
Time for sendtweets 77
Time for queries 523
No of users 1500
Time for create users 16
Time for subscribe 29
Time for sendtweets 128
Time for queries 1585
No of users 2000
Time for create users 32
Time for subscribe 60
Time for sendtweets 206
Time for queries 3038
No of users 3000
Time for create users 59
Time for subscribe 96
Time for sendtweets 412
Time for queries 7258
No of users 5000
Time for create users 107
Time for subscribe 224
Time for sendtweets 1018
Time for queries 20781
```

These are the values obtained when we run the twitter engine.

| No of users | Time to creation of users | Time taken for tweeting | Time taken for querying | Time taken for subscriptions |
|---|---|---|---|---|
| 100 | 1 | 5 | 9 | 1 |
| 200 | 1 | 11 | 33 | 2 |
| 500 | 5 | 40 | 136 | 11 |
| 1000 | 16 | 77 | 523 | 30 |
| 1500 | 16 | 128 | 1585 | 29 |
| 2000 | 32 | 206 | 3038 | 60 |
| 3000 | 59 | 412 | 7258 | 96 |
| 5000 | 107 | 1018 | 20781 | 224 |
| 8000 | 203 | 2284 | 54478 | 594 |
| 10000 | 273 | 3691 | 100159 | 942 |
| 20000 | 865 | 13441 | 355405 | 3706 |

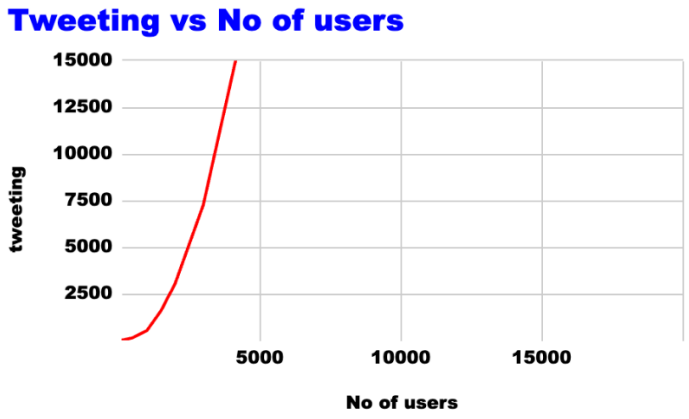
## Graphs:

### Time taken to create users vs Number of users :



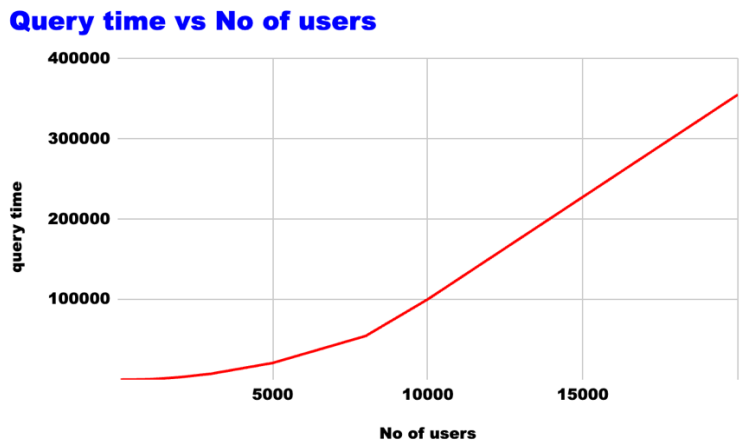

---

### Time taken to Tweet vs Number of users




---

## Time taken for queries vs Number of users:




## Time taken for subscriptions vs Number of users: