COP5615: DOSP Project 4 Part-1 Report

Name: Pranath Reddy Kumbam
UFID: 8512-0977

Problem Statement

The first part of the fourth course project involves the development of a Twitter clone with support for various functionalities and a client tester/simulator. The Twitter engine developed in the first part of the project will be paired up with WebSockets in the second part of the project in order to provide a complete Twitter-like functionality.

language: Erlang

Implementation

As suggested in the problem statement, the implementation has been divided into two separate processes: the client part and the server part. We have a server that distributes the tweets, and multiple independent clients that send and receive tweets working in a single server, multi-client environment. The server and clients communicate over a TCP connection. The clients have information regarding the IP address and the port number that are needed to establish a connection with the server. The server receives and establishes connections with the clients and stores all the information, such as usernames, passwords, tweets, and followers list in ETS (Erlang Term Storage) tables. ETS tables have been used since they are easy to implement, efficient, and fast. ETS table can be easily replaced with a database for future implementations. The implementation supports various functionalities like user registration, tweets, retweets, subscribe, queries, etc, and each functionality has been modeled as a separate service. I have detailed each functionality along with examples in a future section of this report.

Execution Details

For the practical engine and client program, the server requires no input parameters, while the client needs the IP address and the mode of operation. The IP address required to connect to the server and the mode of operation can be “Register” or “Login”. For the simulation program, we require additional parameters. The server takes the total number of users (N), while the client requires the IP address, number of users (N), and a threshold (T).

Program:

%% c(server).
%% server:main().

- *In a different shell:*

%% c(client).
%% client:main(IP Address, "Register").
%% client:main(IP Address, "Login").

Simulation:

%% c(server_simulation).

%% server_simulation:main(N).

- *In a different shell:*
%% c(client_simulation).
%% client_simulation:main(IP Address, N, T).

Functionalities

To register a user, the client can be executed with the "Register" parameter, after which the user is asked to enter their desired username and password. The client then sends the information to the server, which calls the registration_service function, which registers the user by adding the information to the ETS table and sets the user's status to offline. Similarly, to login, the client can be executed with the "Login" parameter. The user is asked to enter their credentials, and the information is sent to the server, which calls the login_service function. This function performs a lookup of the ETS table and sets the user status to "online" if the user has already been registered. This user's connection status will be used for distributing tweets. Similarly, ETS tables are used for storing tweets and follower information as well. The client provides various options for functionalities to the user, such as tweeting, subscribing to a user, retweeting, and querying, such as query via hashtags, fetching tweets that contain mentions of the user, and querying all the subscribed tweets using a keyword. The tweets are delivered live to the users without the need for an explicit request to the server from the user. The functionality options are presented to the user as a persistent service, which means the user can choose an option after each action or after receiving a tweet. This helps avoid the need to restart the client after each action. The user has the option to refresh their feed to display any pending tweets. Examples for various functionalities are presented below.

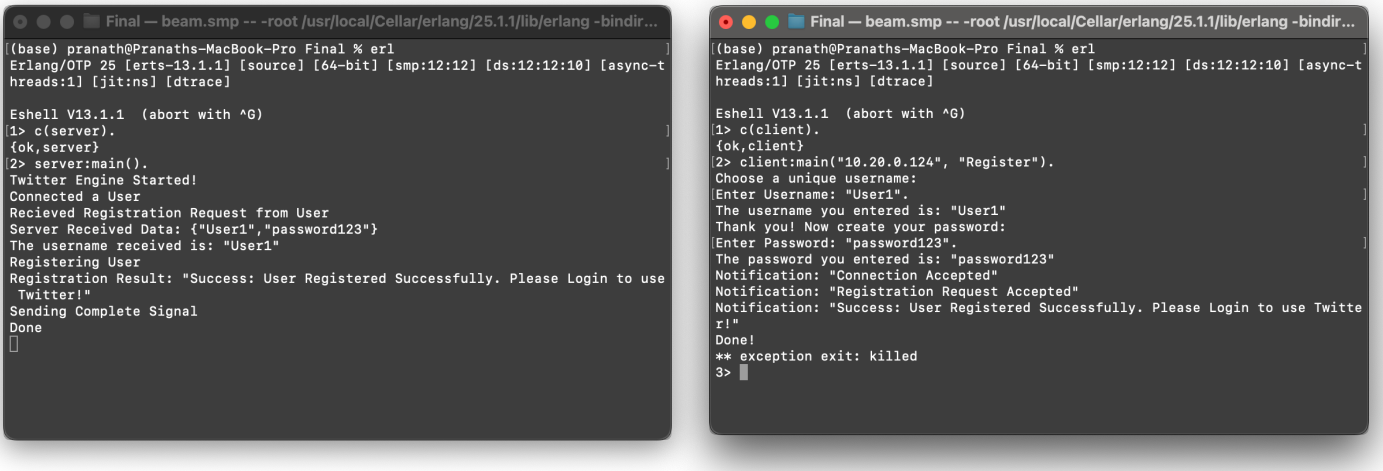

Figure 1: User Registration: Server (Left), Client (Right)

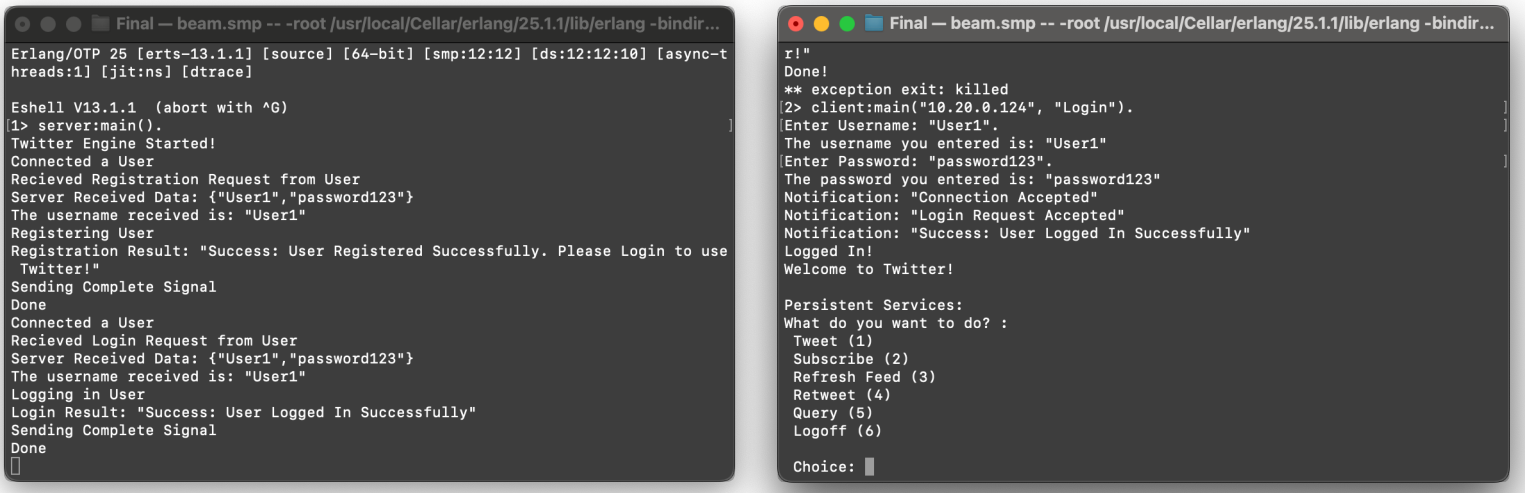

Figure 2: User Login: Server (Left), Client (Right)

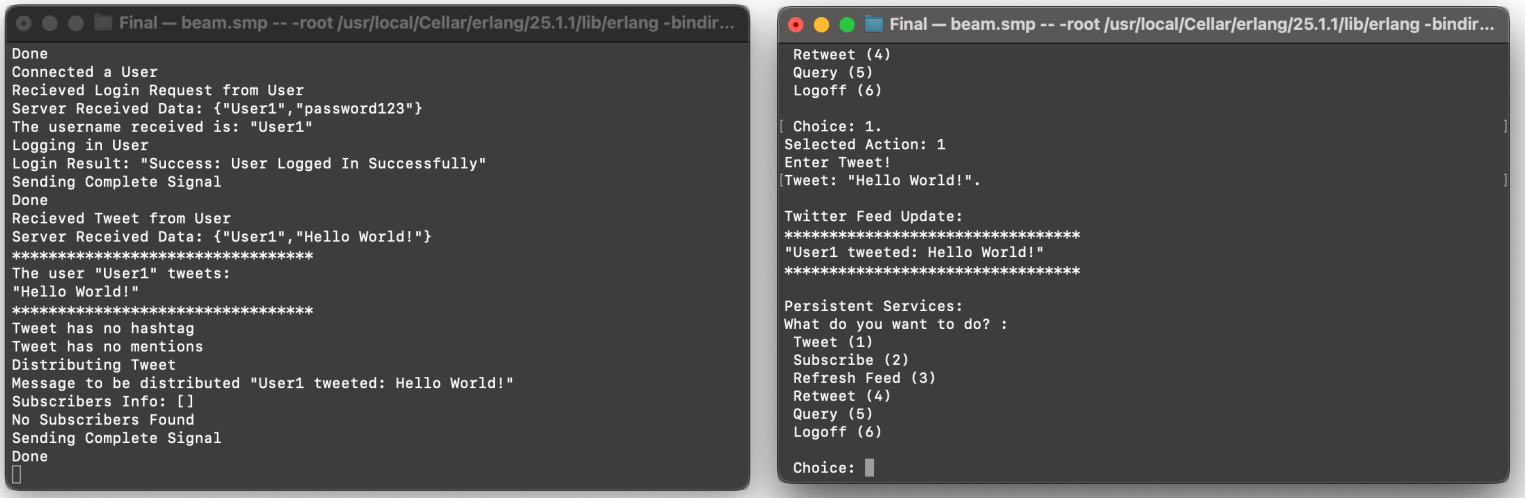

Figure 3: Tweet Functionality: Server (Left), Client (Right)

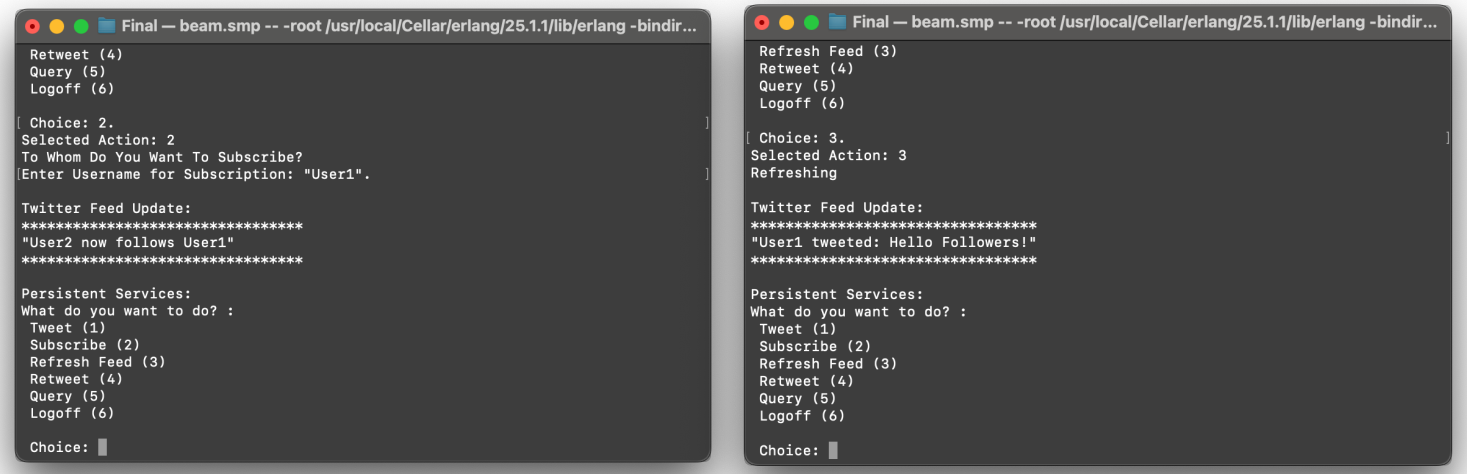

Figure 4: Subscribe Functionality: User 2 follows User 1(Left), User 2 receives a tweet from User 1 (Right)

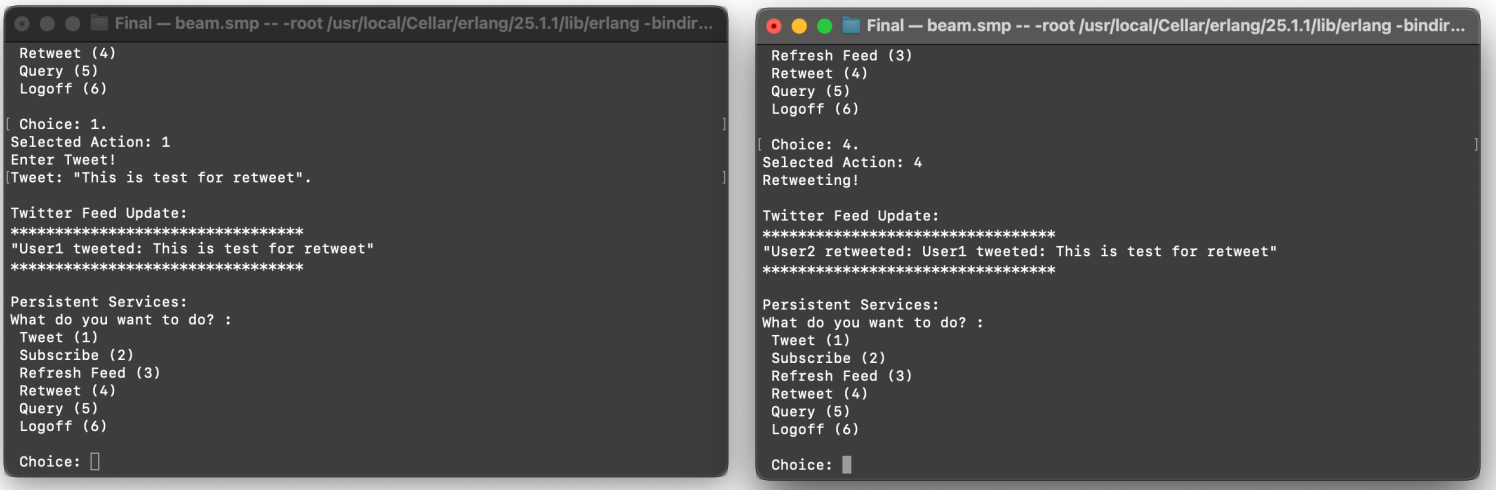

Figure 5: Retweet Functionality: User 1 tweets (Left), User 2 retweets a tweet received from User 1 (Right)

Figure 6: Query Functionality: Query hashtags (Top), Query tweets with specific keywords (Middle), Query my mentions (Bottom)

Submission Details

The submission zip file contains the code, screenshots, simulation results, and plots. server.erl and client.erl correspond to the primary program while server_simulation.erl and client_simulation.erl are for simulation/testing.

Simulation Results

To test the implementation, a simulation is run based on the Zipf distribution. We use Zipf distribution on the number of subscribers and run the simulation for various number of total users. We set a threshold for the number of subscriber and users above the threshold tweet more than those below. Some of the tweets are made into retweets at regular intervals. The users connect and disconnect at regular intervals to simulate a real world environment. We then plot the total runtime and the total number of tweets delivered/distributed. We can see that the runtime is linearly proportional to the total number of users while the number of tweets delivered decrease as we increase the threshold.

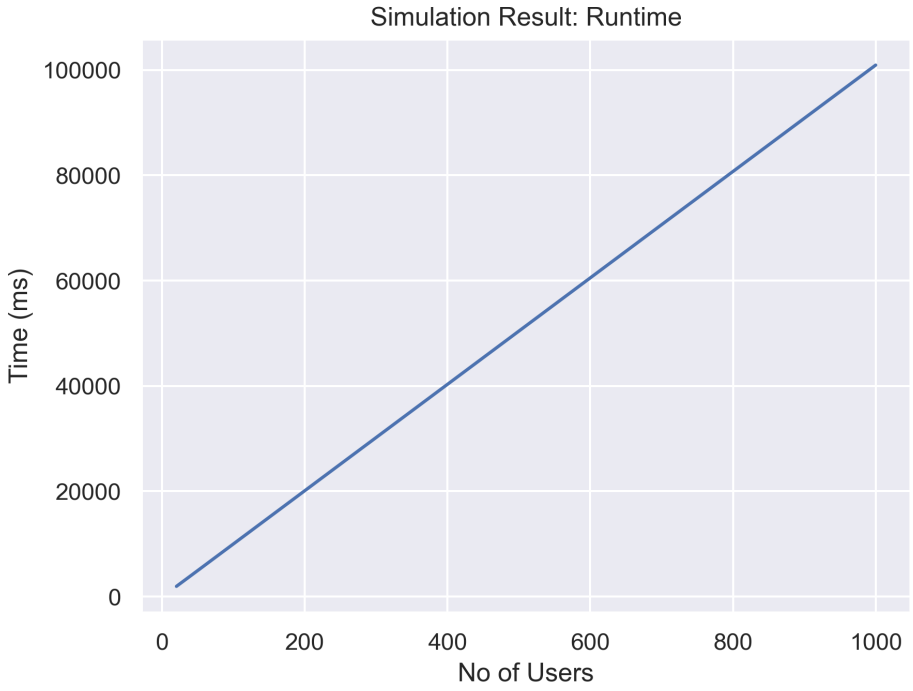

Figure 7: Simulation Time (Time (ms) vs No of Users)

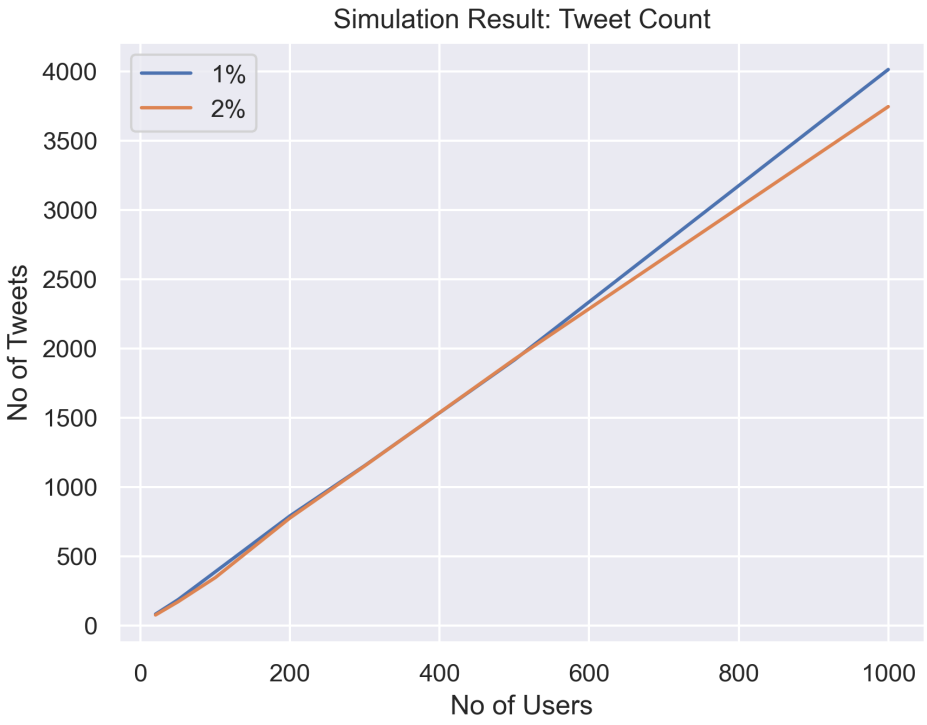

Figure 8: No of tweets delivered (Tweets vs No of Users)