Distributed Operating System Principles : COP5615
Project - 4

Part 2

This project's objective was to develop an **Erlang** Twitter clone along with a client that could test or mimic the server. The 4.1 project section received additional functionality and a structure, and an engine was developed that could grow the 4.1 project. We achieved this by creating a JSON-based API that could represent the messages. I once again used Cowboy to write the code for our engine, creating the **Cowboy** server, and I instructed our client to utilize the web socket.

In this project, we created a server or engine that will eventually be linked to the Cowboy framework in order to provide the full functionality of the now operational Twitter and the database used was **ETS Tables**.

YouTube Link :

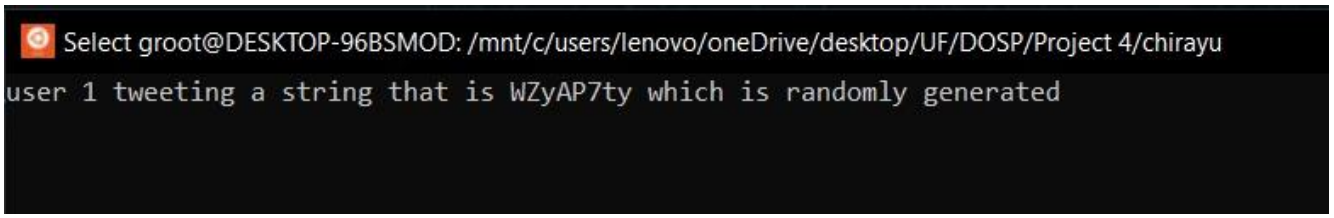
https://www.youtube.com/watch?v=4lJmPzoB99I&ab_channel=VedantJaiswal

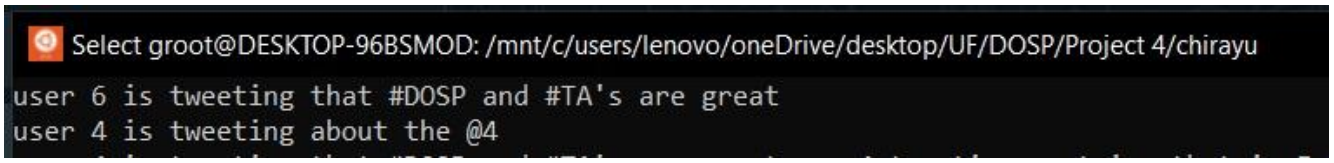
1. Authors :

1) Vedant Jaiswal (UFID: 38827983)
2) Chirayu Tripathi (UFID: 53866363)

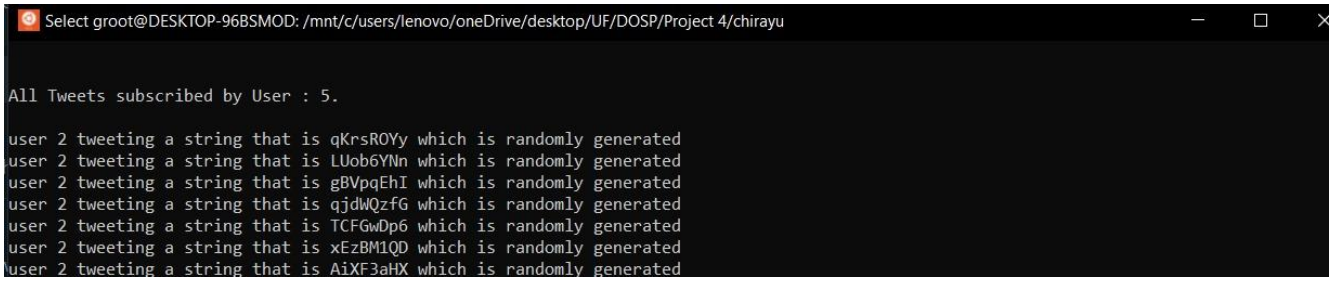
2. Functionality :

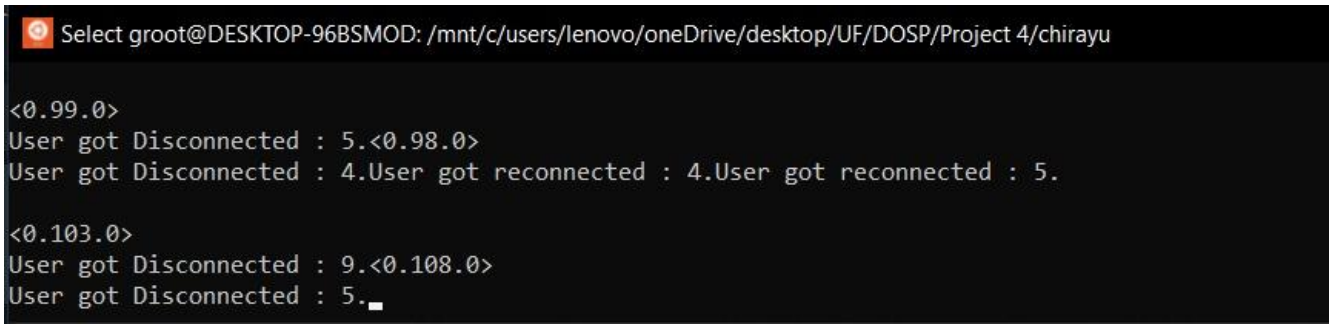
The below added screenshots tell us about the different functionalities our program can perform, it can register an account, send tweets and be able to retweet some other already existing tweet, tweets can also include hashtags for something which is trending and we can also tag a user using the 'mentions' functionality.





A user can also subscribe to some other user's tweets and querying can also be done using the tweets, we can also see the connections and disconnections, that is, the live update of when a user is getting connected or disconnected from our engine using the clone. This is a server-side functionality.





2.1 Zipf Distribution

It gives the client side the most subscribers possible. The client side with the second-highest subscriber count will be followed by the client side with the third-highest subscriber count, and so forth. Utilizing the formula, we determined that total subscribers can be calculated using the number of subscribers and number of clients.

2.2 Client-side

According to how we've built our client, there are 3 parameters: number of customers; most subscribers possible; number of disconnected customers. This file includes information on a single client participant, or a single twitter user. This contains the data associated with its userId, which is stored as a string of numbers given to it during programme starting. The fundamental logic of maintaining process state is carried out by the main registry's ETS table, which keeps track of various actors and is useful for disconnection and reconnection logic. It was necessary to rebuild the client in order to use WebSockets. For websockets, the Cowboy library was used. We had a lot of flexibility since the client was created as a behavior. If the socket has not joined the topic, an error is returned. Upon success, the message reference, also known as the ref, is returned.

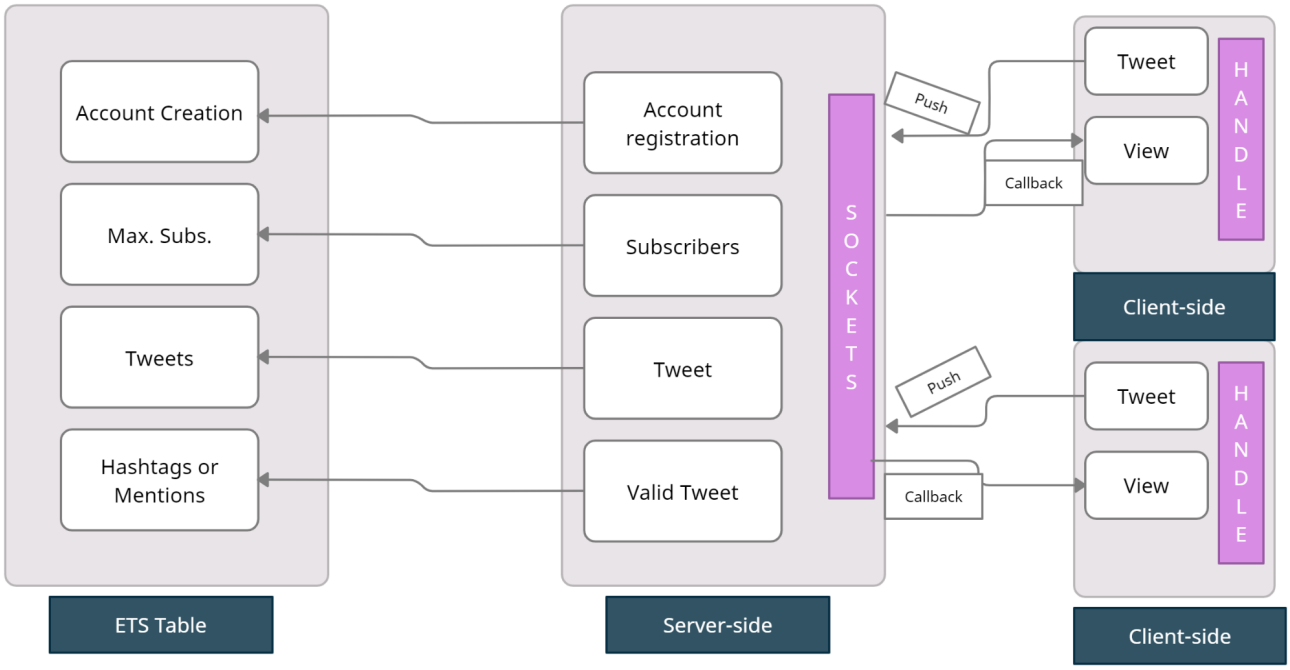
2.3 Server-side

We entirely redesigned our engine in Cowboy in order to accommodate the WebSocket interface. The "example" folder contains the source code for the Twitter engine implementation, which is in charge of processing and distributing tweets. The engine has a direct connection to the database to control things like subscriptions, tweets, and searches (built using ETS tables). It also communicates with users directly using Cowboy channels or websockets to transmit query results, send tweets to subscribers, and do other tasks. Process states are maintained by the client registry ETS table on the server, which keeps track of various actors. Thanks to channels, we were able to rapidly add soft-real time capability to our Twitter application. Channels are built using web sockets. As a transmitter, the server sends messages on a variety of topics. Customers act as

recipients and subscribe to topics in order to get these notifications. Senders and receivers are permitted to switch roles at any time while discussing the same topic. Channel sockets are multiplexed by our Cowboy server over a single connection that it holds. We are able to define default socket assignments for use across all channels as well as authenticate and identify a socket connection thanks to the socket handler modules available in lib/example web/channels/user socket.ex. The preferred mode of transport is WebSockets.

The first criteria of creating a **JSON-based API** was also fulfilled since Cowboy Framework uses JSON internally for data transmission. To confirm this, we looked at cowboy.js, which is automatically included when served from Cowboy.

3. Architecture :

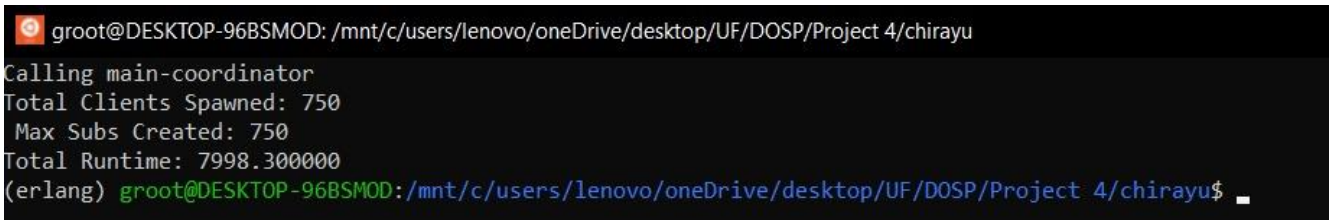


4. Results and Conclusion :

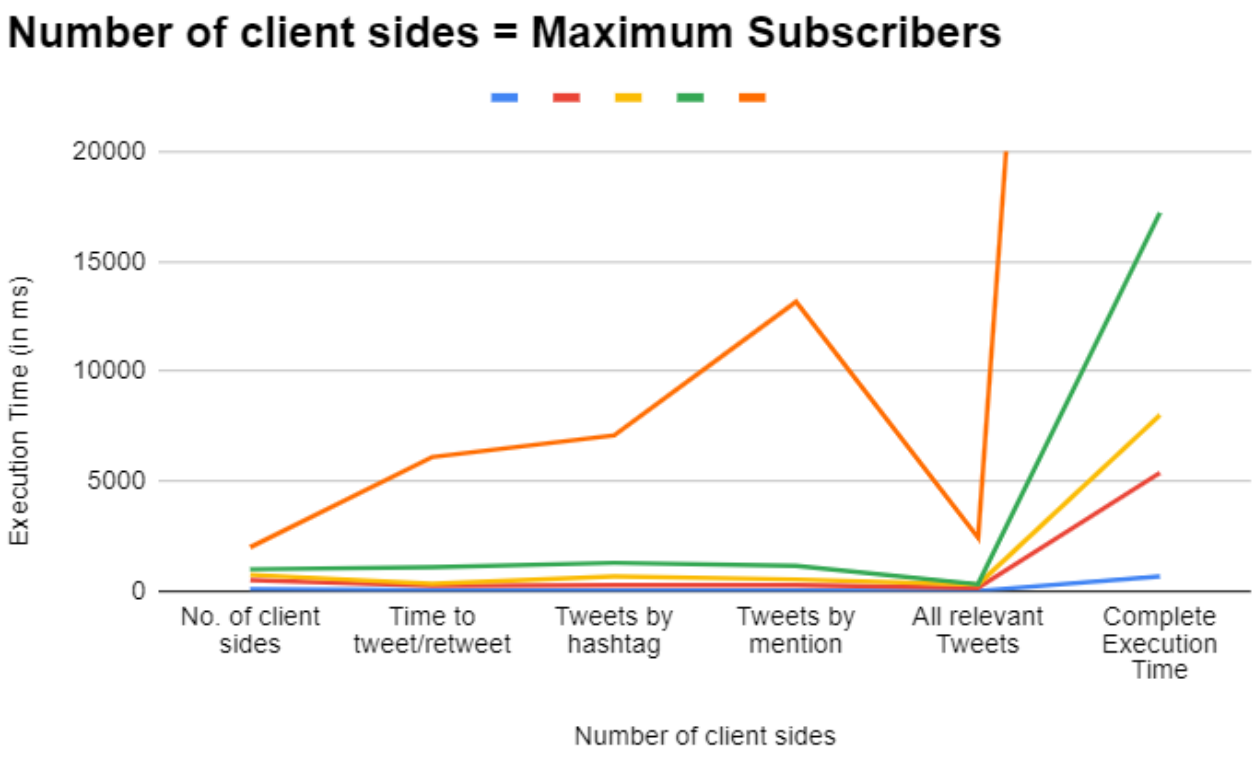
We may assess the results by plotting graphs between the length of time it takes the clone to run and the quantity of client sides. Our programme was able to produce as many client sides as 112301.

Below is a list of the graphs that display the results:

1) Number of client sides = Maximum Subscribers

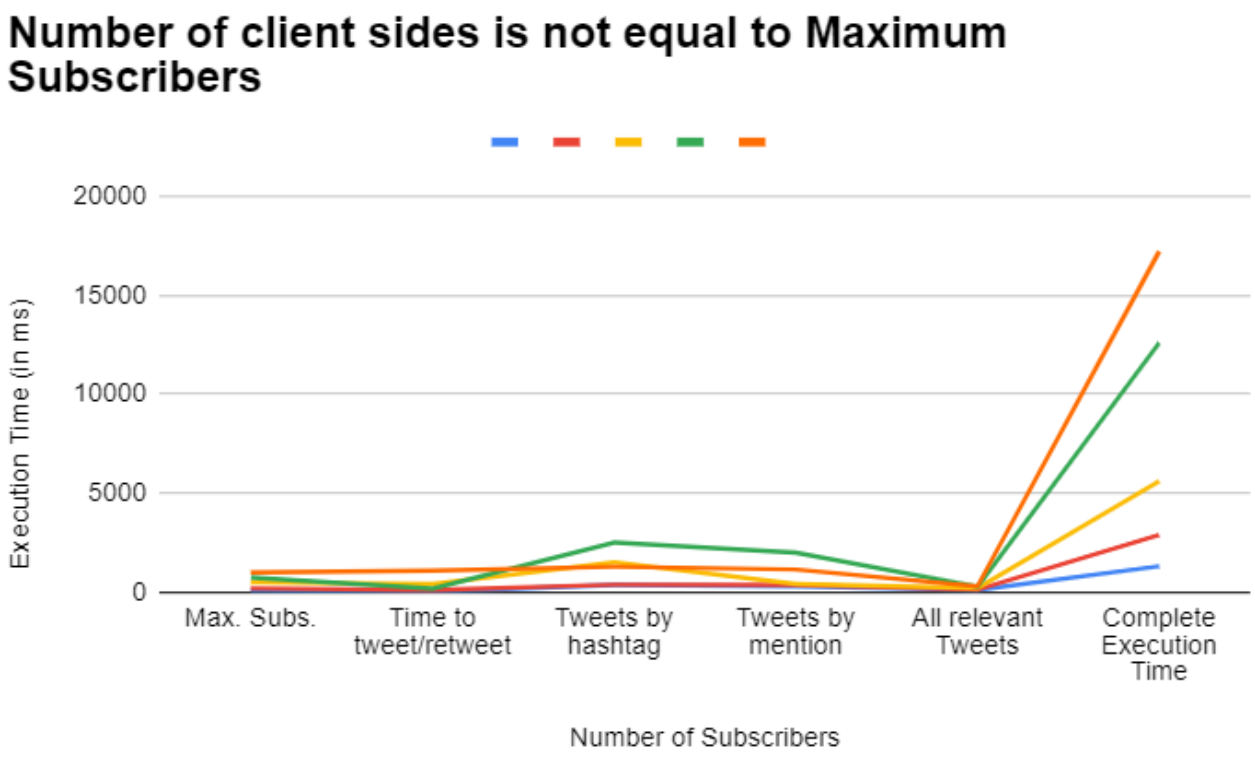


No. of client sides	**Max. Subs.**	**Time to tweet/retweet**	**Tweets by hashtag**	**Tweets by mention**	**All relevant Tweets**	**Complete Execution Time**
100	100	41.02	34.39	37.50	10.90	677.33
500	500	240.85	269.82	280.90	150.80	5372.76
750	750	340.33	677.57	538.68	299.93	7998.30
1000	1000	1091.87	1290.58	1157.20	318.28	17201.65
2000	2000	6100.27	7082.89	13167.48	2434.28	116539.78



2) Number of client sides is not equal to max subscribers

No. of client sides	Max. Subs.	Time to tweet/retweet	Tweets by hashtag	Tweets by mention	All relevant Tweets	Complete Execution Time
1000	100	33.27	378.86	294.28	115.27	1320.96
1000	200	120.66	390.99	370.29	128.27	2900.60
1000	500	415.78	1520.68	420.62	208.34	5600.61
1000	750	186.23	2510.29	2000.37	290.96	12590.37
1000	1000	1091.87	1290.58	1157.20	318.28	17201.65



We were able to build this project in Erlang and utilise the ETS Tables database to record the values of tweets while we tested or replicated the server side or Twitter clone. Cowboy was the framework used to implement web sockets. We created a server side that has a wide range of features, including the capacity to sign up for and create a user account. For the server to identify the hashtag and the person being mentioned, users must add the special symbols "#" and "@" in their tweets. When a user logs in, the server side shows tweets based on their preferences. The user has the option to follow another individual or a hashtag. Since we also generated the client side for this server side, a maximum of 112301 client sides were created. The client side, which now checks the functionality of the different server side functionalities, was launched using the coordinator function. We were able to do the zipf distribution by computing the number of subscribers and tweets on the client side using the calculations mentioned above in this report. This was constructed using distinct procedures for the client and server sides. Distributing tweets is the responsibility of our server-side coordinator, while coordinating their reception and transmission falls to our client-side coordinator.