PROJECT 4 REPORT

1. Twitter Implementation:

Implemented a Twitter clone as a client-server model using Elixir. Server is used to store data and the ETS registry functionality available in Elixir is used for this implementation. Information about registered users, user tweets, retweets, subscription, hashtags, mentions, etc. are stored in respective tables inside ETS registry. Only one server is implemented to handle multiple clients created using actor model. Every client communicates with the server during execution of below mentioned functionalities.

ETS Table	< key, value >	Description
hashTags	< hashtag, tweets >	Hashtags are used as unique identifiers and tweets containing the hashtag are stored as a list.
tweets	< username, tweets >	Username is used as unique identifier and tweets by the user are stored as a list.
user_mention_tweets	< username, tweets >	Username is used as a unique identifier and tweets mentioning the user are stored as a list.
subscriberList	< username, users >	Username is used as a unique identifier and the users subscribed by the user are stored as a list.
subscribersOf	< username, users >	Username is used as a unique identifier and the users following the user are stored as a list.
user_list	< 'regUsers', usernames >	regUsers is the key and all the registered users are stored as a list.

Table 1: ETS registry tables description

1.1 Register account

- Users must go through registration in order to access the twitter implementation.
- Every registered user has a unique username consisting of small a-z, capital A-Z, numbers 0-9, and no special characters.
- Server uses handle_cast during registration to keep track of all the users by storing them in the "*user_list*" table inside ETS registry.

1.2 Delete account

- Users are provided with an option to delete their accounts and server uses handle_call during account deletion, to maintain ETS registry state.
- In case a user deletes his/her account, server is immediately informed, and the corresponding username is removed from *"user_list"* table. This makes the unique username available for future clients.
- Server removes traces of the user's tweets, retweets of these tweets, followers, mentions and subscriptions by updating "*tweets*", "*subscriberList*", *"user_mention_tweets"* tables.
- In the above scenario, corresponding hashtags of the tweet entries are removed from the *"hashTags"* table.
- Server also updates the follower information of those users who have been followed by the deleted account by updating *"subscribersOf"* entries.

1.3 Send tweet

- Users generate random tweets consisting of random strings <sdakmnxdc>, hashtags <#florida> and user mentions <@username>.
- User mentions consists of users tagged in a tweet.
- We are restricting the tweet length to a maximum of 8 words and a minimum of 5 words (can be modified) in our implementation for easy analysis.
- Client forwards generated tweet to server for further operations.
- Server uses handle_cast to add the tweet to the existing list of tweets in the *"tweet"* table corresponding to the user generating the tweet.
- It also adds the tweet to the existing list of tweets in the *"hashTags"* table corresponding to every hashtag mentioned in the tweet.
- It also adds the tweet to the existing list of tweets in the *"user_mention_tweets"* table corresponding to every user mentioned in the tweet.
- Additionally, sends notification to all the followers of the user and users mentioned in the tweet.

1.4 User Subscription

- Users can subscribe to/ follow other users.
- When a user is following other user, server is informed, and it updates the *"subscriberList"* and *"subscribersOf"* tables corresponding to the user.
- Server implementation uses handle_call during account subscription, to maintain updated ETS registry. It is helpful for sending notifications and handling deletion of an account.
- After every tweet by a user, server accesses the above stored *"subscribersOf"* table entries and uses it to send notifications to the user's followers.
- During deletion, server accesses the above stored *"subscriberList"* table entries and uses it to update all the users that the deleted account has subscribed to.
- <BONUS>: As a part of the bonus part, the subscriptions are done using a Zipf distribution. The maximum subscribers possible is varied over a range and reading are obtained for observation. In Zipf distribution the maximum subscribers possible for a user is equal to the (total subscribers/2), the next maximum possible is equal to (total subscribers/3), the pattern follows for the remaining. Observations are presented in section 3.

1.5 Re-tweets

- Users can re-tweet exiting tweets received as notifications.
- In the implementation a user receives two kinds of notifications- via user-mentions (*info = 1*) and subscriptions (*info = 0*).
- User is notified whenever they are mentioned in a tweet. These tweets are available for re-tweet.
- User is notified whenever the users they follow tweet. These tweets are also available for re-tweet.
- Upon re-tweet, the all the users following the user receive notification.
- Server adds the re-tweet to the existing list of tweets in the *"tweet"* table corresponding to the user generating the re-tweet.
- When a user deletes their account, entries of all the re-tweets of their tweets are also removed.

1.6 Querying tweets

- Users can query tweets by other users they are subscribed to, when a query is raised by a user to access the tweets of one the user it is subscribed to, server checks the entry in *"tweet"* table corresponding to the latter's username.
- Users can query tweets by hashtags, when a query is raised by a user to access the tweets containing a specific hashtag, server checks the entry in "*hashTags*" table corresponding the hashtag.
- Users can query tweets in which they are mentioned, when a query is raised by a user to access the tweets mentioning them, server checks the entry in *"user_mention_tweets"* table corresponding the username.

1.7 Notification

- Users when logged in automatically receive notification for tweets/ re-tweets they are mentioned in and tweets/re-tweets posted by the users they are subscribed to.
- Server forwards notifications to mention list and forward list of users after every tweet and re-tweet.
- <BONUS>: A login flag is used to denote the connection of the user. When user is connected, login is set to "1" and disconnection is represented by "0". User receives notification, has access to querying and can tweet/ re-tweet only when connected. The flag implementation handles these scenarios. Additionally, periodic login logout functionality is integrated in the model to replicate real world and test cases are used to verify the effect they have on performance. Observations are presented in section 3.

2. Test Cases Implementation:

The ExUnit framework is used to create testcases. We have implemented multiple unit and functional test cases to test all the twitter functionalities. The verification part is done using assert.

Test 1 & 2: User Registration - These tests add new users and check if they are being registered on the server. For example, when a user is added, there should be an entry in *"user_list"* table corresponding to *"regUsers"*.

Test 3 & 4: User Subscription - These tests check the user subscription functionality, whether the subscriber's data updated by the server matches the user subscription. For example, If A subscribes to B, B should be an entry in *"subscriberList"* table corresponding the username of A and A should be an entry in *"subscribersOf"* table corresponding to the username of B.

Test 5 & 6: Send Tweet - These tests check the tweeting functionality. When a user tweets, the server updates the *"tweets"* table corresponding to the username. In this implementation, length of the entries from the *"tweets"* table for a particular user is cross checked with the input argument *"numMsg"* provided.

Test 7 & 8: Hashtags – These tests check if the tweet containing a hashtag reference is updated in the *"hashTags"* table with reference to the hashtag. This is used while querying tweets with hashtag.

Test 9 & 10: Mentions - These tests check if the tweet containing users mentions is updated in the *"user_mention_tweets"* table with reference to the username. This is used while querying tweets for a particular user mentioned.

Test 11 & 12: Query Tweets <subscribed to> - These tests check if the user can query the tweets of the users followed by them. For example, suppose user A is following user B, when the query tweet initiated by A is successful, the server will access the *"tweets"* table entries corresponding to username of B.

Test 13 & 14: Query Tweets <hashtag> - These tests check if the user can query the tweets containing a hashtag. When the query tweet initiated by the user is successful, the server will access the *"hashTags"* table entries corresponding to the hashtag.

Test 15 & 16: Query Tweets <user mentions> - These tests check if the user can query the tweets containing a user mention. When the query tweet initiated by the user is successful, the server will access the *"user_mention_tweets"* table entries corresponding to the username.

Test 17 & 18: Re-tweet <from subscriptions> - These tests check if the re-tweets match the original tweets. For example, if user A follows user B, then user B tweets are available to user A for re-tweet.

Test 19 & 20: Re-tweet <from mentions> - These tests check if the re-tweets match the original tweets in which the user was mentioned.

Test 21: Testing all the functionalities together - This test will the overall flow of the twitter engine implementation.

Test 22: Delete User – This test checks if all the tables are updated after deletion of an account.

Test 23: Login-logout – This test checks the login and logout functionality. A user only gets tweet/mention notification only when logged in. **<BONUS>**

Test 24: Zipf distribution – This test checks the performance when the user subscriber count follows the zipf distribution. **<BONUS>**

3. Observations

- While implementing twitter clone, few assumptions and approximations have been made which might affect the performance of the model. Since, data is collected over a range of values and multiple runs, a pattern can be obtained from the observations.

Users	Number of Tweets	Time to tweet (ms)
50	50	62
50	100	62
50	150	63
50	200	62
50	250	171
50	300	156
50	500	188
50	750	188
50	1000	157

Table 2: Time taken to tweet when num_user = 50

Users	Number of Tweets	Max subscribers	Time to tweet re-tweet (ms)
50	50	49	290
50	100	49	338
50	150	49	337
50	200	49	359
50	250	49	375
50	300	49	438
50	500	49	390
50	750	49	343
50	1000	49	343

Table 3: Time taken to tweet and re-tweet for zipf subscriber distribution when num_user = 50

Users	Number of Tweets	Time to tweet (ms)
100	50	150
100	100	160
100	150	171
100	200	187
100	250	203
100	300	235
100	500	203
100	750	234
100	1000	248

Table 4: Time taken to tweet when num_user = 100

Users	Number of Tweets	Max subscribers	Time to tweet re-tweet (ms)
100	50	99	4156
100	100	99	4625
100	150	99	4110
100	200	99	4656
100	250	99	4109
100	300	99	4344
100	500	99	4125
100	750	99	4078
100	1000	99	4172

Table 5: Time taken to tweet and re-tweet for zipf subscriber distribution when num_user = 100

- Even with increase in number of tweets per user, time taken to tweet is less compared to subscription time, which can be observed from *Table 2* and *Table 4*. This is happening mainly because, send tweet implementation is using handle_cast, whereas handle_call is used to implement subscription.
- In case where Zipf distribution is used for subscriptions, <maximum subscriptions = total number of users>, <number of re-tweets = number of users>, with increase in number of tweets per user an

increase in tweet time is observed, refer *Table 3* and *Table 5*. This shows that the server takes more time to handle requests when tweets per client are increasing.

Users	Number of Tweets	Max subscribers	Time to tweet re-tweet (ms)
300	50	49	11078
300	100	99	20125
300	150	149	22145
300	200	199	24775
300	250	249	43625
300	300	299	63938

Table 6: Time taken to tweet and re-tweet for zipf subscriber distribution when num_user = 300

- In *Table 6*, Zipf distribution is used for subscriptions, < number of re-tweets = number of users>, when number of total subscriptions and tweets per user are increased, then an increase in tweet time is observed. Time to re-tweet is minimal as number of re-tweets is less than number of tweets. Time taken for subscriptions is affecting the performance. Subscriptions are handled synchronously, so that re-tweets done from subscriptions are consistent.

Users	Number of Tweets	Max subscribers	Time to tweet re-tweet (ms)
500	50	499	400422
500	50	299	116219
500	50	99	18672
500	50	49	12719
500	50	24	9594

Table 7: Time taken to tweet and re-tweet for zipf subscriber distribution when num_user = 500

- In *Table 7*, Zipf distribution is used for subscriptions, < number of re-tweets = number of users>, when number of tweets per user is constant, and number of total subscribers is increased, total time also increases. Time to tweet and re-tweet is still high, comparing these results with *Table 5* results shows that increase in total subscribers increases total time way more than increase in number of tweets. The difference is mainly visible because of the way the two functions are implemented.

Users	Number of Tweets	Max subscribers	Time to tweet re-tweet (ms)
50	50	49	290
100	50	49	2790
300	50	49	11078
500	50	49	12719
1000	50	49	48375

Table 8: Time taken to tweet and re-tweet for zipf subscriber distribution for different num_user

- In *Table 8*, Zipf distribution is used for subscriptions, < number of re-tweets = number of users>, when number of tweets per user is constant, and number of total subscribers is constant, total time also increases with increase in total number of users. Time to tweet and re-tweet is still high, when the total number of users are more, this is because the server's request frequency is increasing.
- *Figure 1* below, shows the effect number of users have over total tweet and re-tweet time.

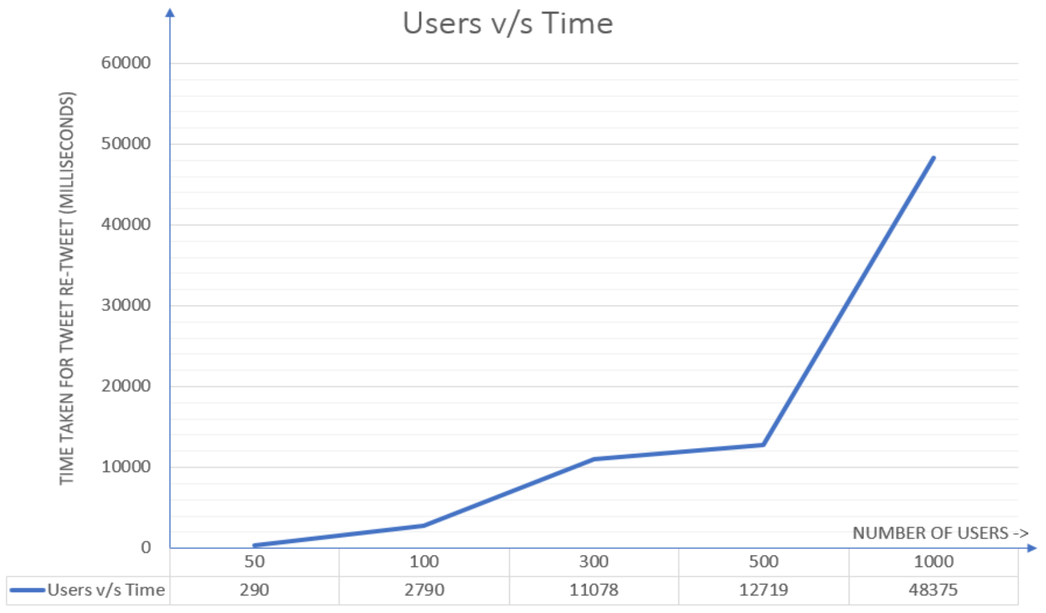

Figure 1: Time taken to tweet and re-tweet for different num_users, when total subscribers = 49, num_msg = 50