PROJECT 4 REPORT

Project Partners:

- Rahul Wahi, UFID – 30536162, rahul.wahi@ufl.edu
- Wins Goyal, UFID – 73571559, winsgoyal@ufl.edu

Implementation:

The basis architecture tree implemented is:

MySupervisor (Boss)→ Clients (Genserver workers) & Server (Genserver worker).

All Client PIDs are assigned to corresponding Users who start creating account on Twitter. The users first register and then login, maintaining their login status for sometime. During when the users are logged in, they can tweet, subscribe, retweet, search, get notifications etc. For the **bonus** part of this project, we are also taking two new parameters from the console: PercentageOfDisconnection, maxNumberOfSubscribers.

Following tables are used to store the data in ETS registry. We have implemented different tables for client side and server side. Table name, schema, and description of the tables are as follows.

Table Name	Schema	Description
notification	user, [tweet, tweet_owner, index]	This table stores the notifications for a user, which can be accessed by key = user.
client_tweet	user, [user, tweets]	This table stores tweets by a user, which can be accessed by key = user.

Table1: Tables in ETS registry on client side

Table Name	Schema	Description
user	user, [user, password, login_flag, pid]	This table stores the user details – username, password, login_flag, pid. It can be accessed by key = user.

tweet	user, [user, tweets]	This table stores the tweets by a user, which can be accessed by using key = user.
retweet	user, [tweet_owner, tweet_index]	This table stores the re-tweet information of a user – tweet owner and the tweet index. It can be accessed by key = user.
subscribe	user, [subscribe_by_user, susbscribe_to_user]	This table stores the user subscription information – user, user subscribed to. It can be accessed by key = user.
pending_notification	user [tweet, tweet_owner, index]	This table stores the pending notifications for s user. It contains tweet, tweet owner and index. It can be accessed by key = user.
mention	user, [mentioned_tweets]	This table stores the tweets with user mentions. It can be accessed by key = user.
hashtag	hashtag, [hashtag, tweets]	This table stores the tweets containing particular jashtag. It can be accessed by key = hashtag.

Table2: Tables in ETS registry on server side

Functionalities:

- **User registration:** This functionality is implemented to register the users on the twitter server. Every user is assigned a unique username. All the usernames are updated in the "user" table in the ETS registry.

- **User Login/ Logout:** This functionality allows the users to login and logout. Every user has unique login credentials – username and password. The username and password are stored in the "user" table in the ETS registry.

- **Tweet:** This functionality is used by the clients(users) to tweet. The tweets done by all the users are updated on the twitter server in the "tweet" table on the server.

- **Re-tweet:** This functionality allows the user to re-tweet a tweet. User can re-tweet any tweets from notification. The re-tweets are stored in "retweet" table.

- **Subscribe:** This functionality allows the user to subscribe to another user. The information is stored in the "subscribe" table.

- **Live notifications:** This functionality is used to send live notifications to the user if the user is logged in. If the user is not logged in, the notifications are stored in the "pending_notification" table and are sent to the user once user is logged in. All the notifications are stored in the "notification" table.

- **Querying:** This functionality allows the user to query tweets by following:
    - **My mentions –** A user can query the tweets in which he/ she is mentioned
    - **By hashtag –** A user can query tweets containing particular hashtag.
    - **Subscribers tweet –**A user can query the tweets of the users he/ she is subscribed to.

- **Periodic Login/ Logout:** Implemented a periodic login logout functionality which changes the login state of the user after a random time period.

- **ZipF distribution:** The number of subscribers users can have is based on zipf distribution. The total number of subscribers/ users to be subscribed is distributed to each users according to the the distribution function. E.g. Because the ZipF is similar to that of an expontential dropping function, the distribution of number of subscribers decreases exponentially such that very few users have highest number of subscribers. Hence, some users will have large number of subscribers, while some will have least number of subscribers. And, we try to simulate the performance of our Engine according to this real world scenario.

  We implement this in the Simulator by using the following formulation:
  To get the ZipF distribution:→ *Kernel.trunc(maxSubscribers/ user) +1*

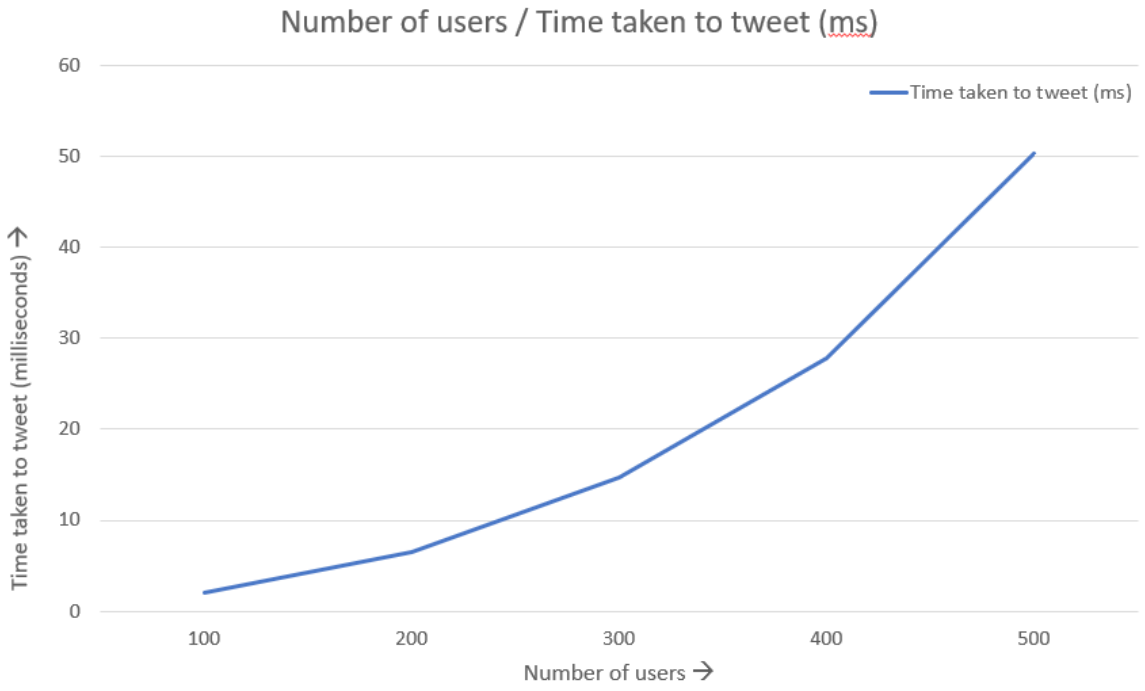
Test Cases:

- **Test Case 1:** Creating 1 twitter client and verifying if it is created
- **Test Case 2:** Creating 1000 twitter clients and verifying if they are created
- **Test Case 3:** Verifying the server ets tables creation – subscriber table
- **Test Case 4:** Verifying the server ets tables creation – hashtags table
- **Test Case 5:** User registration
- **Test Case 6:** Multiple users' registration
- **Test Case 7:** Verify that users with duplicate names cannot be created
- **Test Case 8:** Check for Incorrect login credentials – incorrect username
- **Test Case 9:** Check for Incorrect login credentials – incorrect password
- **Test Case 10:** User login - Testing successful login
- **Test Case 11:** User login and logout

- **Test Case 12:** User login/ logout – multiple users
- **Test Case 13:** 1000 users' login
- **Test Case 14:** Tweeting, one tweet per user
- **Test Case 15:** Tweeting, one tweet with hashtag per user
- **Test Case 16:** Tweeting, with mention
- **Test Case 17:** Tweeting, subscription and notification verification
- **Test Case 18:** Logout 1000 users
- **Test Case 19:** Query by mentions/ user
- **Test Case 20:** Query by hashtag
- **Test Case 21:** Deactivate users

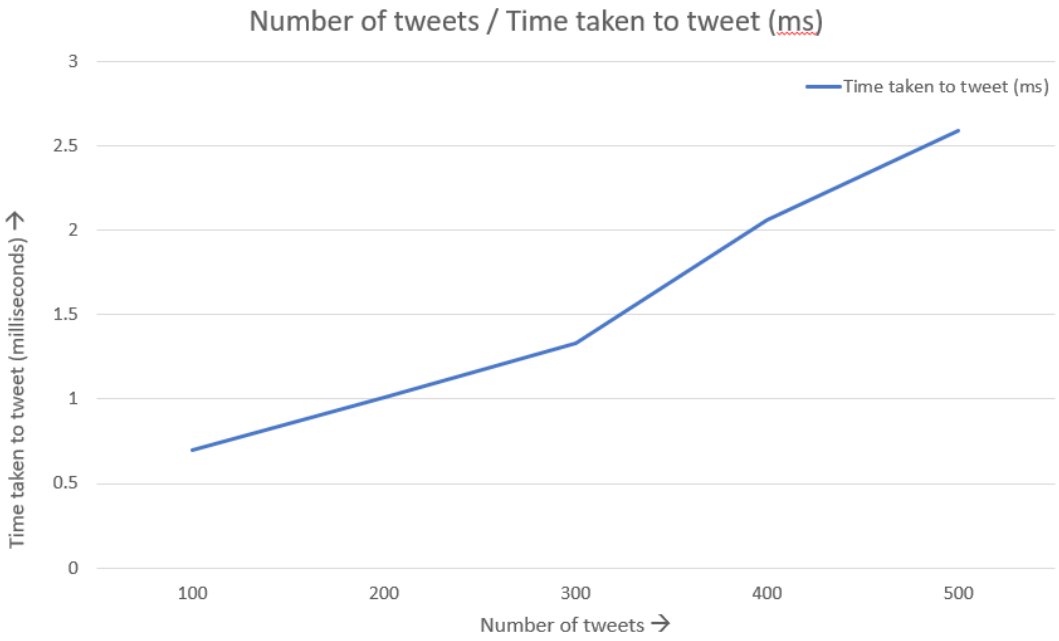
Performance (Observations):

1. There is a exponential increase in the time taken to tweet by users when number of users increase, while keeping the number of tweets by each user constant. E.g. here, we keep the number of tweets by each user as 10.



Graph1: Time taken to tweet by users when number of tweets by each user = 10

2. While, if we keep the number of users constant, there is linear increase in the time taken to tweet by users when the number of tweets per user increase.



Graph2: Time taken to tweet by users when number of users = 10

3. Hence, it seems that number of users is more significant factor than the number of tweets in finding out the performance of the Engine simulator.