PROJECT 4 – II: TWITTER SIMULATOR

Implementation Overview:

- The client and server are implemented in two different projects.
- Server side is implemented using phoenix framework. Thus, making use of WebSockets.
- Client side is making use of phoenix_gen_socket_client which enables use of WebSockets.
- While sending and receiving messages, phoenix framework internally converts the messages to
- JSON format. Hence, it doesn't need to be done explicitly.
- The individual functionalities of client and server side are same as that in the previous project.

Steps to execute:

1) To start server:
   i. Install dependencies with `mix deps.get`
   ii. Install Node.js dependencies with `cd assets && npm install`
   iii. Start Phoenix endpoint with `mix phx.server` in server location

2) To start client:
   i. Install dependencies with `mix deps.get`
   ii. Compile using `mix escript.build`
   iii. Start client using ` escript client “No of clients” “No of requests sent by client” `

Implemented Functionalities

Client side:

Each client can perform the following functionalities:

1) Register: Before sending any request, each client has to register with the server.

2) Subscribe: Each can subscribe to other clients. While implementing subscription we have maintained the Zipf distribution i.e. more popular clients have more subscribers and less popular clients have less subscribers.

3) Tweet: Each client can send tweets relative to the number of tweets provided as the input. These tweets maintain the Zipf distribution i.e. more popular clients tweet more. Tweets may contain randomly generated user mentions (identified by @) and hashtags (identified by #).

4) Retweet: Each client can retweet some of the messages received by it. Retweets are sent to server just like usual tweets except that they have “rt:” added in the beginning of the tweet content. The frequency of the retweet is also based on Zipf distribution. The more popular the client the more it will retweet.

5) Query my mentions: Each client can query for tweets they are mentioned in. On querying they will get the 100 latest tweets they are mentioned in.

6) Query hashtags: Each client can query random hashtags. On querying they will get the 100 latest tweets that has those hashtags.

7) Disconnect: Each client can disconnect from Twitter. When it disconnects, it no longer receives live tweets. Also, it can't query for its mentions or hashtags. While simulating periods of disconnection we have maintained the Zipf distribution i.e. more popular clients are disconnected for shorter amount of time.

8) Connect: Once disconnected the client can connect again. As soon as the client re-connects it's gets all the tweets that were done while it was disconnected. After this it can resume with its usual activities.

Terminating condition: We ask no of tweets as user input. This indicates number of tweets that should be sent by most popular users. The number of tweets sent by other users is calculated accordingly. Once the desired number of tweets are sent. The client process stops.

Server functionalities:

The basic functionality of the server is to process requests of the clients. In order to maintain data, it makes use of ETS tables. The following tables are present in the server:

A) user_details: This table stores user id and whether the user is currently connected or not.
B) subscriber_details: This table maintains the subscribers of each user.
C) hashtag_table: This table maintains 100 latest tweets of every hashtag ever tweeted.
D) mentions_table: This table maintains 100 latest tweets of every user mention ever tweeted.
E) wait_table: In case a user is not connected, the tweets for that user are stored in this table.

Once the user gets connected all the tweets present in this table are forwarded to the user and this table is cleared.

Based on the type of the request the server processes it as follows:

1) Register: When a user registers an entry for that user is created in the user_details table.
2) Subscribe: When a user, say, A ask to subscribe to another user, say, B, an entry for user A of made with key user B in the subscriber table.
3) Tweet: Each user can tweet messages. When a tweet request is received by the server it does the following:
    a. Parses for Hashtags: Check if the tweet has any hashtag. If yes, then add the tweet in the hashtag table along with the hashtag as the key. Also, as we are maintaining only 100 latest tweets for each hashtag. If the table already has more than 100 tweets/hashtags it will delete the oldest tweet and then insert the newest tweet.
    b. Parses for user mentions: Check if the tweet has any user mentions. If yes, then add the tweet in the user mentions table along with the mentioned user as the key. Just like in hashtag table, since we are maintaining only 100 latest tweets for each mentioned user. If the table already has more than 100 tweets/user, it will delete the oldest tweet and then insert the newest tweet.
    c. Forward tweets: On receiving the tweet the server checks for subscribers of the user in the subscriber_details. For all the subscribers of the user, it needs to forward the received

tweets. Before forwarding it to the subscriber it checks whether the subscriber is connected. If yes, then forwards it. If no, then it adds tweet to the wait table.

4) Query my mentions: When the server receives query for user mentions, it searches in the user_mentions table if there is any entry for the said user. If yes, then it sends all the corresponding tweets.
5) Query hashtags: When the server receives query for hashtags, it searches in the hashtag table if there is any entry for the said user. If yes, then it sends all the corresponding tweets.
6) Disconnect: When the server receives a disconnect message from the client, it changes the "is connected" in the user details table to "N".
7) Reconnect: When the server receives a reconnect message from the client, it changes the "is connected" in the user details table to "Y". It then checks in the waiting_table if there are any messages for the user. If yes, then it sends all the message to the user.

Terminating condition: There is no terminating condition for the server it has to be terminated explicitly.

Zipf Distribution

To implement Zipf implementation we have maintained 5 types of clients. The popularity of each client varies. Depending on the popularity of the client they send more tweets. The distribution of the clients are as follows:
Type 1: 5% (Most popular)
Type 2: 10%
Type 3: 15%
Type 4: 25%
Type 5: 45% (Least popular)
In order to maintain the zipf distribution, the "number of clients" parameter is rounded off to the nearest 100th value. The more popular clients tweet more often and are disconnected for a lesser amount of time.

References:

1) https://github.com/Aircloak/phoenix_gen_socket_client