Project Report: Distributed Operating System Principles (COP5615)

Project 4 (Part 2): Twitter Engine and Client Simulator using websockets

Name: Vikas Chaubey**, UFID:** 3511 5826**, Email:** vikas.chaubey@ufl.edu

1) **Server-Side Functionality** (Server.Fsx File): All the back-end functionality of the twitter Engine is defined in the server.fsx file. All these endpoints are created using websockets. Websockets are implemented using **Suave library**. All endpoints and their functionality as per specification given in the problem statement are as below:

1) **RegisterUser Endpoint** : This endpoint takes registration request from the user and register the user in the twitter engine. In order to register with twitter engine, user needs a valid username and password, this username is unique to every user. The user records are updated in data tables on registration request, and the server sends and acknowledgement message to user to client side about success or failure of registration.

2) **Login Endpoint:** This endpoint lets user to login in the twitter system and start a session to do twitter activities such as tweeting and following other users etc. In order to login a user needs to send username and password to the server, this user record is matched with records present in data tables, if user record match is successful then user login becomes successful otherwise login fails. An acknowledgement message is shared with user to client side about success or failure of Login.

3) **Follow Endpoint:** This endpoint lets user to follow other twitter users. When a user subscribes or follow other user on twitter system then he receives the tweets made by followed user in his twitter feed. This endpoint performs following operations:

   a) When user follows another user, the user is added to the followers list of followed people. In follower's data table.
   b) When user follows another user, the followed user is added to the Subscription list of current users. In subscription data table.
   c) An acknowledgement message is shared with user to client side about success or failure of Follow operation.

4) **Tweet Endpoint:** This endpoint lets logged in users to tweet.an user can use mentions(@user4) and hashtags(#art) in their tweets. When a user makes a tweet following operations are performed at backend:
   a) The Tweet is added to tweet list of users which keeps all users tweet record in tweet data table
   b) The tweet is added to the feed of all users who follow current user, by updating feed table of all subscribers. And the tweet is sent in the feed of all active followers, so followers can read it or retweet it as per their wish.
   c) All the mentions in the tweet are discovered with a regular expression function and if any mention is found then this tweet is added in the mention table, mention string as key and tweet as record. This is done so that all the tweets with same mentions could be queried later.

d) All the Hashtags in the tweet are discovered with a regular expression function and if any hashtag is found then this tweet is added in the hashtag table, hashtag string as key and tweet as record. This is done so that all the tweets with same hashtags could be queried later.

e) An acknowledgement message is shared with user to client side about success or failure of Tweet operation.

5) **Retweet Endpoint:** This Endpoint lets user to retweet any tweet made by other followed user. When a user retweets a specific tweet, then this retweet is sent to all the subscribers of user who are in the followers list. When a retweet is made original author is mentioned so that wherever the tweet is retweeted original author is always identified.

6) **HashTagQuery Enpoint:** This endpoint takes a request to query all tweets in the system which have a specific hashtag (ex #crypto). This is done by querying data tables which are designed to keep track of all hashtags and related tweets. An acknowledgement message is shared with user to client side about success or failure of **HashTagQuery** operation.

7) **MentionQuery Endpoint:** This endpoint takes a request to query all tweets in the system which have a specific mention (ex. @user2). This is done by querying data tables which are designed to keep track of all mentions and related tweets. An acknowledgement message is shared with user to client side about success or failure of **MentionQuery** operation.

8) **GetAllUserTweets Endpoint:** This endpoint lets user to query all the tweets made by a specific user. This is done by querying the tweet table which contains all the records of all users in the system.

Other Server Details:

1) Server side implements a remote actor active on localhost on 9000 port.
2) All the data tables used on server side are thread safe.
3) The project run 128 instances of server actor in order to handle maximum number of user request and simulates load balancing behavior of actual server-side model

Server Run command: **(dotnet fsi --langversion:preview server.fsx)**

All the server-side responses are sent to client side and printed on client side.

2) **Client-Side Simulator and Performance tester – (Client. fsx):** The client simulates the real users and simulates the whole twitter environment registration, login, sessions, logouts, tweeting, following, retweeting among the simulated users. It also tests the performance of the server-side endpoints by measuring the endpoint response times under heavy load when the simulation runs. Client-side endpoints and functionality is below:

1) The client.fsx file is run after the server is started on different port in different process in terminal **2)** Client boots at IP: 0.0.0.0 and port 8080

3) Client is run using the command: **dotnet fsi --langversion:preview client.fsx (UserCount)**
4) The user count simulates the given number of users in the system
5) For Given number of users a Zipf distribution of followers is created, the user with high rank has more followers than a user with low rank. A user with high rank tweets more, the tweets distribution is also done with ZipF distribution.

ZipF Distribution:

Occurrence = (ZipF constant * Total Number of Elements) / (Rank of the Element)

Here ZipF constant is considered to be 0.30 for both tweet and subscriber distribution

6) Once the Distribution is done, to simulate user Actors are created, each actor simulates an actual user.
7) Actor objects are created and each actor is associated with an use rid, password, subscription list(users that actor will follow during simulation) , tweet rate (the rate at which user will tweet as per its rank in zipf Distribution) , session time (time interval between login and logout session)
8) Once the user is created, it contacts server to register itself with the twitter engine. If the server acknowledges the successful registration then user proceeds, in case of failure user tries to register again with different use rid.
9) After Registration, user logins in the system, to login into the system user contacts server and send its username and password, if the match is found in the user database then server allows user to login otherwise login is failed
10) On successful login, the user session is started now user can do twitter activities. This session is active for fixed interval of time example 120 seconds. All twitter activities will be done by user in this time.
11) To simulate twitter functionality, user starts following other users which are already decided by the zipf distribution, user sends follow request to server to follow all other user present in its subscription list. Server sends acknowledgement on successful follow operation.
12) When user completes follow operations for all its subscription list, then it starts tweeting, every tweet has a randomly picked mention (among all users) and hashtag (from 10 popular twitter hashtags) and a general message. User send the tweet request to server and server distributes this tweet among all followers of user.
13) User receives tweets from its subscribed users in its feed, it can randomly choose to retweet it or ignore it. Client simulates that behavior, when retweet decision is made a retweet request is sent to the server. Server distributes this retweet among all the followers of the user.
14) The user makes the hashtag query, mention query and user tweets query on server

15) When the session is timed out user sends a logout request to the server and server removes it from active user list and sends an acknowledgement on the status of logout operations
16) When all user's logout from the system, then recorded performance parameters are printed.
17) All the server responses on success and failure of any operations are printed on the client side.

3) **Performance of The System:** The performance of the system is measured while the client is running simulation for given number of users. The basic idea behind testing performance is to test the response time of different server endpoints when the number of users is increased. During simulation the average roundtrip time between a client request and server response is measured to evaluate how the endpoints perform when under heavy load. Performance measure is done in Blocks which are below:
   a) **Maximum Load:** How many users can system handle without any failures. The maximum number of load that this system could handle with 32 instances of server running on 8 core I9 processor is **900 users,** when the number of users were increased to 1000 then actors got stuck and messages were getting lost hence client was not receiving the acknowledgements from the server hence operations were stopped. **Performance Metrics: System is tested to run maximum 900 users successfully. The maximum number of simulated requests are 3536 for session(login -logout time duration) timeout period of 300 seconds (5 minutes).**

   b) **Performance table for different loads:** On each completion of simulation evaluation parameters are printed, this table evaluates the performance based on those parameters. Each simulation is run on a 8 core 16 gb machine.

Number of simulated users	**Session duration For each user (in seconds)**	**Total registration request made**	**Total login request made**	**Total follow request made**	**Total tweet/retweet request made**	**Total logout request made**	**Total data Query Requests made**
20	60	20	20	17	142	19	20
100	120	100	100	87	456	98	48
300	180	300	300	265	1671	300	178
500	240	500	500	398	2678	498	234
900	300	900	900	789	3536	854	423

Succesful Requests for which response was received from server:

Number of simulated users	Session durationFor each user (in seconds)	Succesful registration request made	Succesful login request made	Succesful follow request made	Succesful tweet/retweet request made	Succesful logout request made	Succesful data Query Requests made
20	60	20	20	11	139	19	20
100	120	100	100	67	430	87	39
300	180	300	289	211	1599	231	156
500	240	500	487	298	1421	452	197
900	300	900	846	656	3025	765	357