Distributed Operating System Principles (COP5615)

Project-4 Part 2

Zhaoyang Chen (UFID: 9013-7235)

1. Project Overview

Use WebSharper framework to implement the web-socket interface to the functionality of Part 1. Use AKKA messaging framework to build Twitter engine. The server actor has been modified to use web-socket and the client actor will send messages to the web-socket to realize asynchronous communication. All messages, feedbacks as well as error will be passed based on the JSON API.

2. Functionalities

Register account: Take userID and Password as inputs and add new users to the network and update (modify follower table and number of users, etc.). Duplicated userID is not allowed, error raised.

Log out: User log out from the twitter and will be removed from the online user set.

Log in: For the registered users, they can log in the engine with userID and password. Handle the errors if password is wrong, the user is not registered, duplication login.

Send tweet: Take userID and the content of tweets as inputs. Store the tweets with hashtags or with mentions.

Add follower: Subscribe a user. The user can view other's tweet only after the subscription.

Re-tweet: The user can randomly re tweet a tweet based on the given userID

Query all mentions: Display all the tweets in which the users have been mentioned.

Query all hashtags: Display all the tweets that include the hashtags.

Query all tweets: Display all the tweets a user has subscribed to, i.e. His/her own tweets and the tweets from people he/she is following.

3. Implementation

1. Design JSON based API.

Use the module *JsonSerializer* to serialize and deserialize the JSON data. From the server side, sending the response for each action and from client side, fetching the data through deserialization.

2. Implement web-socket interface using WebSharper

```
//instantiating websocket server
let wssv = WebSocketServer("ws://localhost:9001")
// build websocket receive handlers
.
.
// Add handlers to web socket service
wssv.AddWebSocketService<Connect> ("/Register")
 wssv.AddWebSocketService<LoginUs> ("/Login")
 wssv.AddWebSocketService<LogoutUs> ("/Logout")
 wssv.AddWebSocketService<Subs> ("/Subscribe")
 wssv.AddWebSocketService<Twet> ("/Tweet")
 wssv.AddWebSocketService<QueryMention> ("/QueryMentions")
 wssv.AddWebSocketService<QueryHasH> ("/QueryHashTags")
 wssv.AddWebSocketService<Disconnect> ("/Disconnect")
 wssv.AddWebSocketService<Retweet> ("/Retweet")
 wssv.AddWebSocketService<QueryTweets> ("/QueryTweets")
 wssv.Start ()
```

3. Client re-write

Based on erlang code from Part 1, use the web-socket and WebSharper framework to build the client architecture. We can simulate multiple clients by spawning client actors.

4. How to run

1. Modules used:

a. Akka

b. System.Text.Json

c. WebSocketSharp.Server

2. **Environment:**
    a. Microsoft Visual Studio Community 2022 (64-bit) – Version 17.3.3
    b. Windows 10
3. **Run**
    a. Under the directory ***server***, open terminal run the following:
        i. dotnet build
        ii. dotnet run
    b. Same under the directory ***client***, open terminal run the following:
        i. dotnet build
        ii. dotnet run
4. **Demo Link**

https://youtu.be/oCu5Rd0c-XM