Deep Bodra
5801 1841

README

TEAM MEMBERS

Individual Project

INSTALL DEPENDENCIES

```
dotnet add package Akka.Remote
dotnet add package Akka.Serialization.Hyperion
```

RUNNING THE CODE

Run server.fsx

```
dotnet fsi --langversion:preview server.fsx
```

Run client.fsx

```
dotnet fsi --langversion:preview client.fsx numberOfClients
```

WHAT IS WORKING

- Everything :)
- Account Creation
- Login
- Tweet
- Retweet
- Search Tweets
- Live Tweets (send tweets in real time to active users)
- Statistics

IMPLEMENTATION

1. Client Simulator
    a. This acts as a local actor
    b. The parent first creates N users (actors) and then assigns each user as one of these
        i. Aggressive Reader
            1. 10% of the users are of this type (According to the twitter statistics)
            2. They log out with a probability of 0.05 and login with the probability of 0.95
            3. If already logged in then they query for tweets with the probability of 0.80 and post a tweet with the probability of 0.2
            4. They take an action every 5-10 seconds
        ii. Aggressive Publisher
            1. 10% of the users are of this type (According to the twitter statistics)
            2. They log out with a probability of 0.05 and login with the probability of 0.95

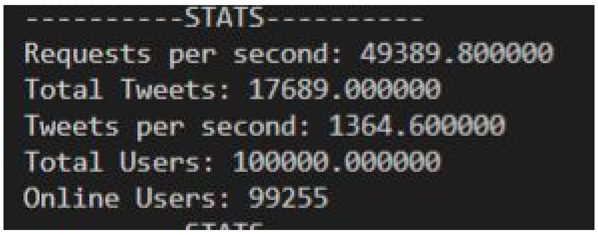
            3. If already logged in then they query for tweets with the probability of 0.20 but post a tweet with the probability of 0.8
            4. They take an action every 5-10 seconds
        iii. Lazy
            1. 80% of the users are of this type (According to the twitter statistics)
            2. They log out with a probability of 0.3 and login with the probability of 0.70
            3. If already logged in then they query for tweets with the probability of 0.35 and post a tweet with the probability of 0.35
            4. They take an action every 10-15 seconds
    c. Random number are generated to decide the next action and when to take the next action
2. Server
    a. This acts as a remote actor
    b. Different hashmaps are used that acts as in memory database
    c. Server responds to different message types by modifying the hashmaps
    d. Information about the user is stored so that the tweets can be sent to the client in real time

PERFORMANCE

For 100K clients

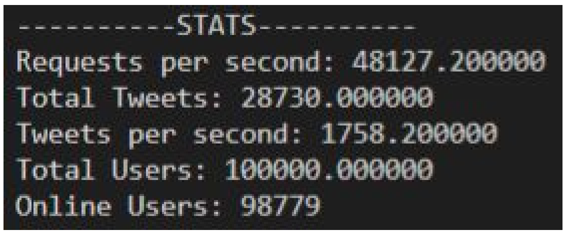
It will take some time to reach this number because initially everyone is just joining. After everyone has joined, the users start doing other things like Tweeting, querying, etc

1. Maximum Requests per second (49389.8)



```
----------STATS----------
Requests per second: 49389.800000
Total Tweets: 17689.000000
Tweets per second: 1364.600000
Total Users: 100000.000000
Online Users: 99255
```

2. Maximum Tweets per second (1758.2)
   This number is low because only 10% of the users are active at a given time and they don't tweet at the same time since I have introduced randomness in the client behavior



```
----------STATS----------
Requests per second: 48127.200000
Total Tweets: 28730.000000
Tweets per second: 1758.200000
Total Users: 100000.000000
Online Users: 98779
```

3. Maximum Users Served

100K. I have not tried more but the code will be able to handle as many users as possible

CONCLUSION

We used Actor model to implement Twitter Server and also simulated different client behaviors to see the performance of our engine