**DOS Project 4 Part I**

Aniketh Sukhtankar (UF ID 7819 9584)

Shikha Mehta (UF ID 4851 9256)

**Brief Description**:

The goal of this project is to implement a Twitter Clone and a client tester/simulator. The problem statement is to implement an engine that (in part II) will be paired up with WebSockets to provide full functionality. The client part (send/receive tweets) and the engine (distribute tweets) were simulated in separate OS processes. Multiple independent client processes that represented up to 10 million simulated clients were spawned during an iteration and handled by a single server process.

The Twitter engine (Server) has been implemented with the following functionality:

* Register account
* Send tweet. Tweets can have hashtags (e.g. #COP5615isgreat) and mentions (@bestuser)
* Subscribe to user's tweets
* Re-tweets (so that your subscribers get an interesting tweet you got by other means)
* Allow querying tweets subscribed to, tweets with specific hashtags, tweets in which the user is mentioned (my mentions)
* If the user is connected, deliver the above types of tweets live (without querying)

Architecture

Send Tweets

H

A

N

D

L

E

R

Clients registry

Request

A

P

I

H

A

N

D

L

E

R

Register User

**Implementation Details**:

Response

Response

Request

Live View

Query

Send Tweets

Live View

Query

Datastore (ETS Tables)

H

A

N

D

L

E

R

Twitter Client

Twitter Client

Twitter Engine

Query Handling

Get/Set Subs

Get/Set Follow

Process Tweet

Insert Tags

Followers

Tags/Mentions

Tweets

Subscribers

Main.ex: This elixir file is the entry point and hosts the main method. Our implementation takes 3 parameters in the order - numClients, maxSubscribers, disconnectClients

* numClients: the number of clients to simulate (eg. 100)
* maxSubscribers: the maximum number of subscribers a Twitter account can have in the simulation (eg. 50)
* disconnectClients: the percentage of clients to disconnect to simulate periods of live connection and disconnection (eg. 10)

Server.ex: This file contains the code for Twitter engine implementation that is responsible for processing and distributing tweets. The engine directly communicates with the database (implemented using ETS tables) to handle subscribers, tweets, queries, etc. It also communicates directly with clients via the API handler to distribute the tweets to subscribers and send the query results. Inside the server maintaining process states is done by the clientsregistry ETS table that keeps track of various actors.

Client.ex: This file consists of the information pertaining to a single client participant that stands for a single twitter user. This includes the information of its userId which is stored as a numeric string passed to it upon initiation. The underlying logic of maintaining process state is done by the mainregistry ETS table that keeps track of various actors and is useful for disconnection and reconnection logic.

**Zipf distribution** - As per the requirements we were asked to simulate a Zipf distribution on the number of subscribers. The Zipf distribution was handled by taking an additional parameter from the user, maxSubscribers which provides the maximum number of subscribers a client can have in the simulation. The client with the second highest number of subscribers had maxSubscribers/2 total subscribers while the client with the third highest number of subscribers had maxSubscribers/3 total subscribers and so on. This was achieved by calculating the number of clients a user should subscribe to in order to achieve zipf distribution using the formula noToSubscribe = round(Float.floor(totalSubscribers/(noOfClients-count+1))) – 1, where ‘count’ is the userId of a client. Every client then subscribed to noToSubscribe other clients thereby achieving the required Zipf distribution. Further, we also increased the number of tweets

For accounts with a lot of subscribers the number of tweets had to be increased so as to increase load on the engine for tweet distribution. This was done by ensuring that for any user account, its total tweets are equal to the number of its subscribers, using the formula round(Float.floor(totalSubscribers/count)) for calculating noOfTweets.

**Periods of live connection and disconnection for users** – The third parameter taken by the program is disconnect clients which takes in the percentage of clients to disconnect to simulate periods of live connection and disconnection. If <disconnectClients> parameter is 0, the client’s simulator console displays the performance statistics at the end. Otherwise, it prints the statistics and continues to simulate recurring periods of live connection and disconnection.

Re-tweets are handled by makingevery client randomly pick a tweet from one of its subscribers and retweet it to its own subscribers. A “-RT” is appended to the end of a retweet to differentiate it from normal tweets as in original Twitter.

All the queries with tweets subscribed to, tweets with specific hashtags, tweets in which the user is mentioned (my mentions) were handled successfully by message passing to the server and displaying the list of tweets received on the client side. The tweets will be delivered live to a user that is online by means of live view. User ID is prefixed to this output to identify which user's live view is getting updated.

**Performance Results**:

The time taken for the simulation to run vs the number of clients is plotted as a graph. A maximum of **116738 clients** have been spawned on our system. Since our client server implementation is completely distributed, we are confident that more clients can be spawned with suitable availability of CPU power. The values and consolidated graphs for results we obtained have been disclosed below:

Performance when Max Subscribers(and max tweets/account) = Number of Clients (0% disconnectClients)

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| Number of Clients | Max  Subscribers | Time to Tweet, Retweet  (milliseconds) | Query Tweets Subscribed To  (milliseconds) | Query Tweets by Hashtag  (milliseconds) | Query Tweets by Mention  (milliseconds) | Query All Relevant Tweets  (milliseconds) | Complete  Execution  (milliseconds) |
| 100 | 99 | 43.96 | 122.83 | 36.52 | 15.83 | 6.42 | 734 |
| 500 | 499 | 253.39 | 2309.92 | 320.418 | 333.19 | 146.386 | 5109 |
| 750 | 749 | 301.61 | 3523.74 | 633.11 | 418.57 | 206.92 | 7047 |
| 1000 | 999 | 1256.88 | 6616.81 | 1294.74 | 703.46 | 197.14 | 16531 |
| 2000 | 1999 | 5855.21 | 54364.68 | 7393.04 | 17024.85 | 2494.94 | 115141 |

Performance for different number of Max Subscribers / Max Tweets per Account (0% disconnectClients)

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| Number of Clients | Max  Subscribers / Max Tweets per Account | Time to Tweet, Retweet  (milliseconds) | Query Tweets Subscribed To  (milliseconds) | Query Tweets by Hashtag  (milliseconds) | Query Tweets by Mention  (milliseconds) | Query All Relevant Tweets  (milliseconds) | Complete  Execution  (milliseconds) |
| 1000 | 100 | 36.16 | 172.65 | 364.73 | 311.63 | 167.23 | 1734 |
| 1000 | 200 | 100.00 | 380.17 | 377.56 | 335.57 | 111.49 | 2094 |
| 1000 | 500 | 412.78 | 1468.85 | 1311.19 | 404.98 | 160.67 | 5688 |
| 1000 | 750 | 139.92 | 7047.60 | 2999.58 | 1700.60 | 271.67 | 13484 |
| 1000 | 999 | 1256.88 | 6616.81 | 1294.74 | 703.46 | 197.14 | 16531 |

For our system, a decrease was observed in the maximum number of clients supported by our engine when the number of tweets per account was increased, i.e. by increasing the maxSubscribers parameter.

**References:**

* http://highscalability.com/blog/2013/7/8/the-architecture-twitter-uses-to-deal-with-150m-active-users.html
* https://elixirschool.com/en/lessons/specifics/ets/
* https://medium.com/@Stephanbv/elixir-phoenix-build-a-simple-chat-room-7f20ee8e8f9c
* https://elixir-lang.org/