Distributed Operating System Principles (COP5615)

Project-4

Group Members

JAYACHANDRA GUNTURU (UFID: 56276000) **SATYA NAGA AKHILESH IRRINKI (UFID: 64730546)**

---

1. Twitter Engine:

Implement a Twitter-like engine using Akka.net Actor model. The Twitter Engine should support the functionality of Registering accounts, Sending Tweets, Subscribing to Tweets, Re-Tweet and Querying Tweets. The tweets can have Hashtags and user mentions. Querying would be done for subscribed tweets, tweets with specific hashtags, tweets in which the user is mentioned (my mentions).

2. Twitter Simulator:

The simulator should simulate users with periods of live connection and disconnection. The number of subscribers for each user should be in accordance with Zipf distribution. The account with more subscribers should be making more tweets and retweets.

3. Considerations:

The client part (send/receive tweets) and the server (twitter clone engine) must be in separate processes. Preferably, need to use multiple independent client processes that simulate thousands of clients and a single-engine process.

4. Implementation:

Implemented the Twitter engine and Twitter Simulator as two different processes.

I. Twitter Engine (Server):

- The Twitter Engine is our server, and it allows users to Register, Subscribe, Tweet, Retweet, Querying, Hashtag, mention other users, log in and log out.
- Each service is handled by different actors i.e., for every action there is a unique actor, and it takes care of the corresponding request.

- The request handler is the main actor at the server. It that takes all the incoming requests and redirects them to the respective actors.
- The printer actor prints the total number of requests processed. We have set the printer actor to print the total number of requests processed so far for every 5 seconds.
- A user's subscribers list, tweets, retweets, mentions and hashtags are stored at the server.
- We have encrypted the tweets using the symmetric key encryption at the client side. Therefore, the tweets stay encrypted at the server. They are only decrypted back on the client side.

II. Twitter Client (Simulator)

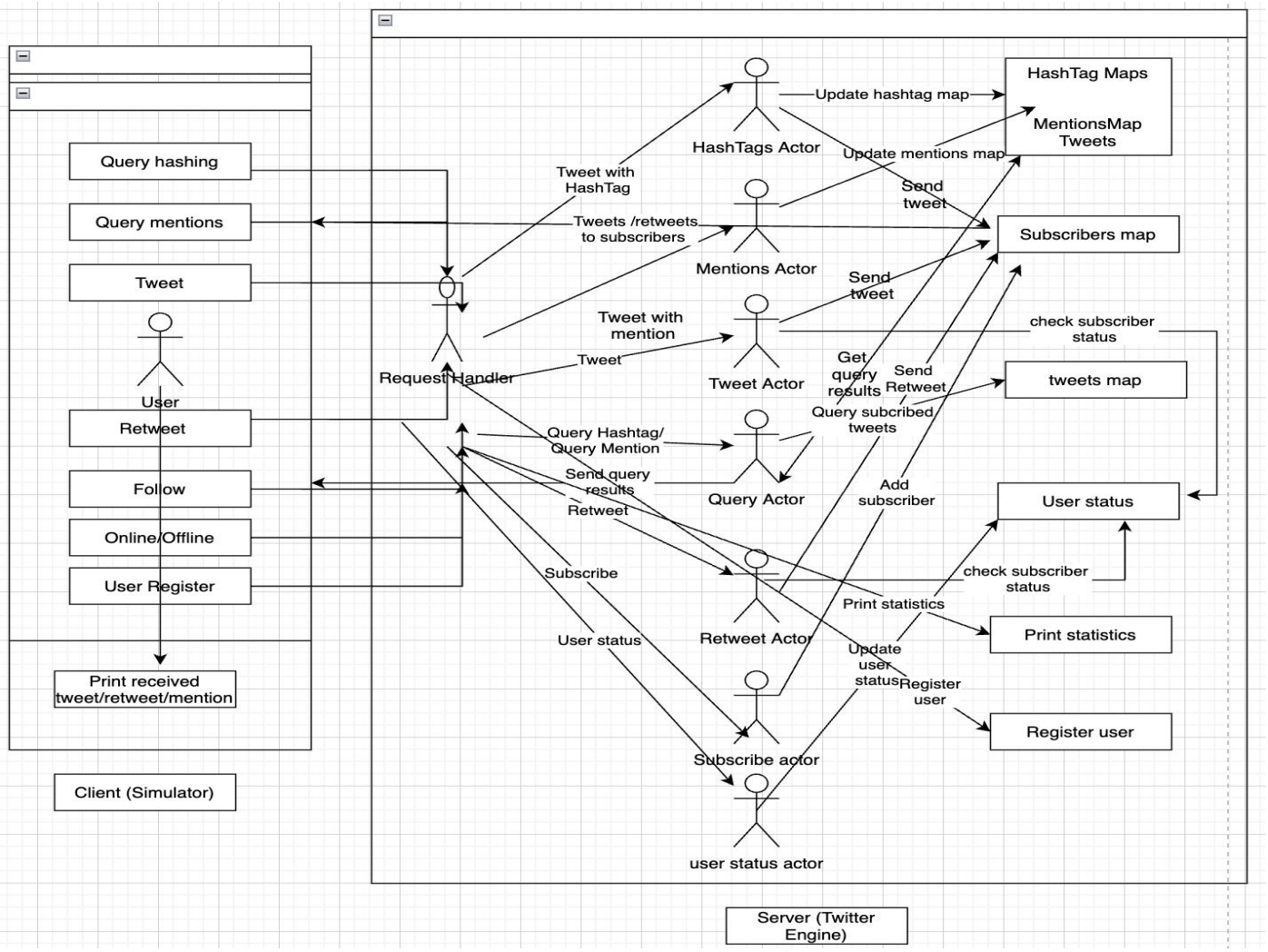
- The twitter client first spawns the given number of users.
- As a user cannot perform any action without registering all the users are first registered.
- Next each user performs random actions i.e., the user might tweet, retweet, subscribe to another user, mention other user, logoff, login, query for mentioned tweets, query for tweets with specific hashtags and search for subscribed tweets.
- For tweeting and retweeting we have pre-loaded some of the hashtags and tweet messages at the client side. A user would randomly select a hashtag or a tweet message when it needs to perform a tweet, retweet or to mention another user.
- Multiple clients can be used for simulating multiple users. The starting id should be distinct from one another for all clients.
- As the tweets are encrypted and had to decrypted on the client side while printing it the printer takes some time to print the tweets.

5. Zipf Distribution:

- We have implemented Zipf distribution on the number of subscribers.
- Zipf distribution is based on the rank of the user. As we are storing the user references in an array, we have used the array index of the current user as the rank as it is unique.
- Suppose there are 100 users to be simulated, the user with the first rank (first index) will be having 99 subscribers, the user with the second rank (second index) will be having 49 subscribers, the user with the third rank (third index) will be having 33 subscribers and so on the subscribers are allocated for all users.
- The same is applied even if multiple clients have been connected. The Zipf distribution would be applied individually for all the clients.
- For example, if 5 clients have been connected and each had simulated 100 users each, then there would be 5 users for each rank with the same number of subscribers thereby following the Zipf distribution.

- Coming to the action's frequency performed by each user, we have assigned time interval for each user. A user performs a random action after every time interval that is assigned. For example, if an user is assigned an time interval of 1ms, the user would be performing actions after every 1 ms.
- We have assigned time intervals for each user using a strategy like the zipf distribution.
- The user with the most number of subscribers will be having very small time interval and the user with the least number of subscribers would be having a bit large time interval.
- Suppose there are 100 users then, and subscribers have been assigned to all the users according to zipf distribution. Then the user with most subscribers i.e., the user with first rank having 99 subscribers will be having a time interval of 1, followed by the user with second rank with the same time interval and the last ranked user would be having a time interval of 99 ms.
- Thus, the user with the most number of subscribers would be performing actions i.e. tweeting, retweeting, hashtag and mention more frequently than the user with low number of subscribers.

6. Architecture of Simulator and Twitter Clone Engine



7. Commands to run the program:

- Dependencies: We have used Hyperion serializer for passing Types from Client (simulator) to the Server directly across the Network. Please ensure that Hyperion has been installed.
- The Zip file consists of Server.fsx (Twitter Engine) and Client.fsx (Simulator files).
- In the Client.fsx and Server.fsx files please change the hostname in the configurations to the IP address of the machines in which the files are opened.
- Commands for starting the server,

dotnet fsi server.fsx

- Command for starting the client,

dotnet fsi client.fsx "ServerIP" noOfUsers UserStartID

- For multiple clients to simulate users please ensure that UserStartID's are distinct for each client.
- For example, if we must simulate 1500 users with three clients, the commands for starting multiple clients are,

First client terminal - dotnet fsi client.fsx "ServerIP" 500 1

Second client terminal - dotnet fsi client.fsx "ServerIP" 500 2

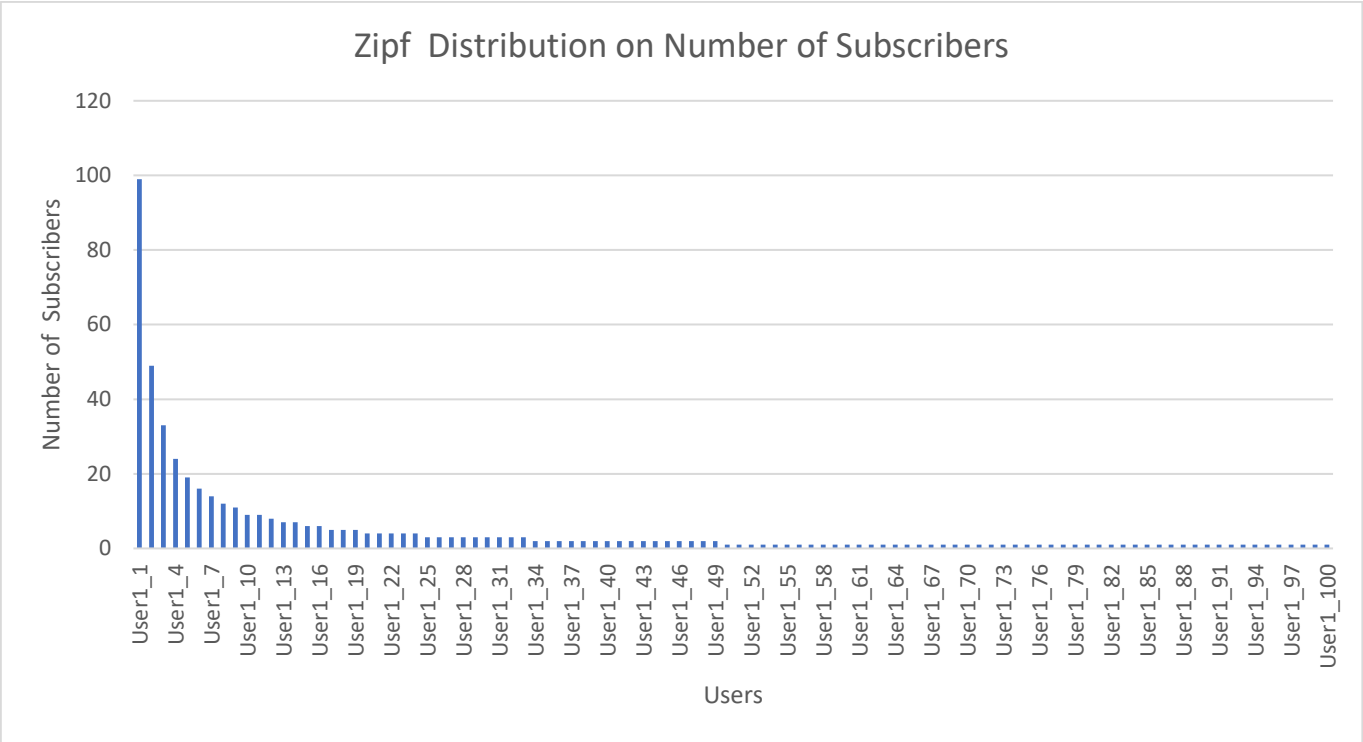
Third Client Terminal- dotnet fsi client.fsx "ServerIP" 500 3

8. Reason for using distinct start ids for multiple clients:

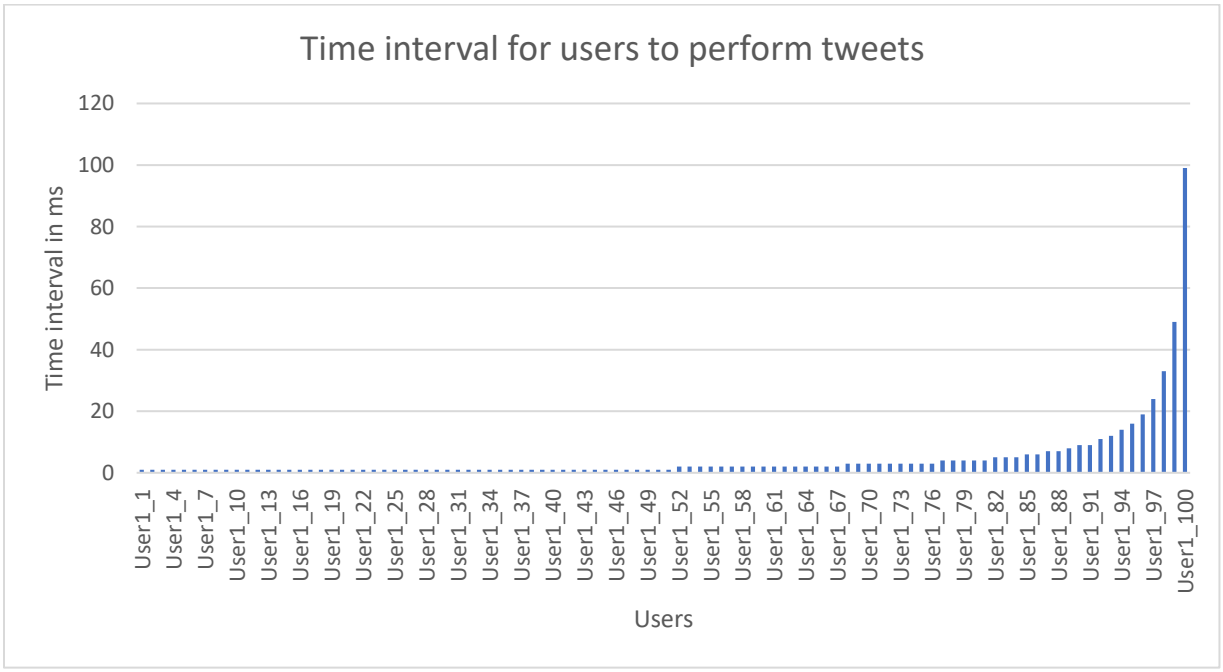
- We are creating usernames as follows, User_someNumber. So, if 100 users are to be created the usernames would be User_1, User_2, ……, User_99, User_100.
- So, if we try to simulate users from multiple clients there would be repetition in the usernames, and it would affect the other user's data.
- To avoid clashes between usernames a distinct starting id is to be used. It helps to generate unique usernames.
- That is the reason a distinct starting id is to be given when running multiple clients.

9. Zipf Distribution on number of Subscribers:

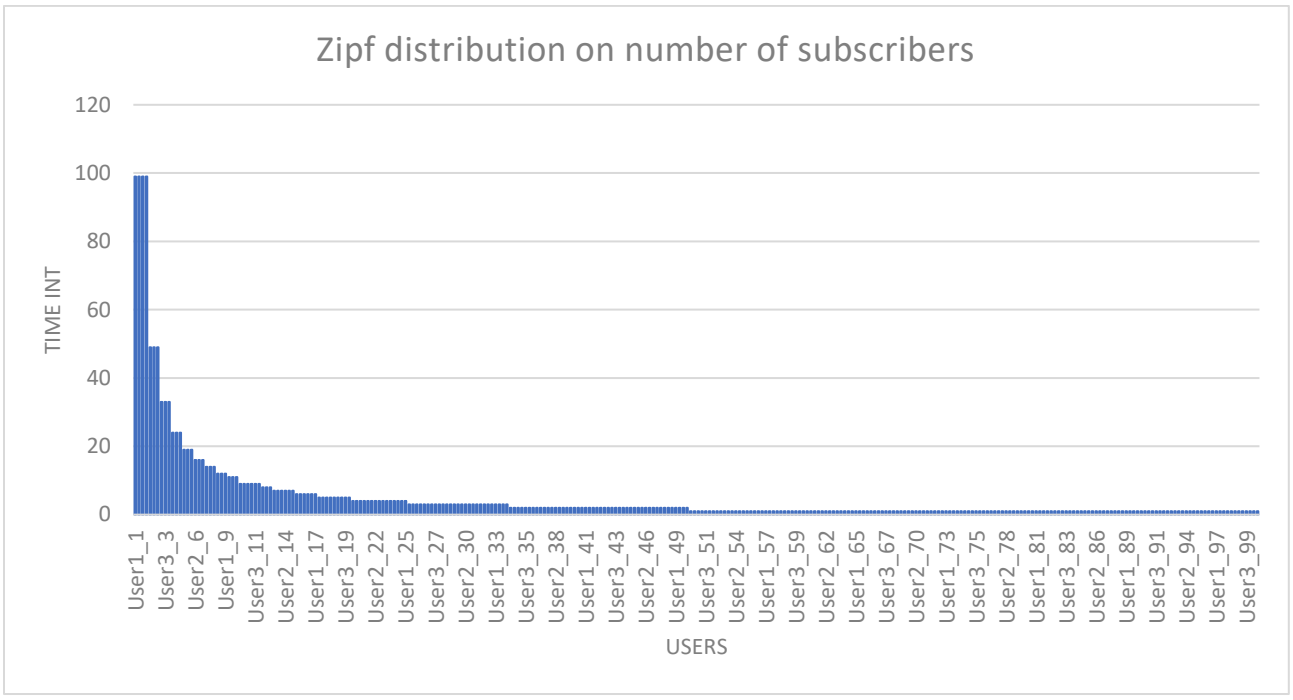
i. Zipf distribution on number of subscribers for each user ranging from 1 to 100 (**With 1 client and 100 users**):



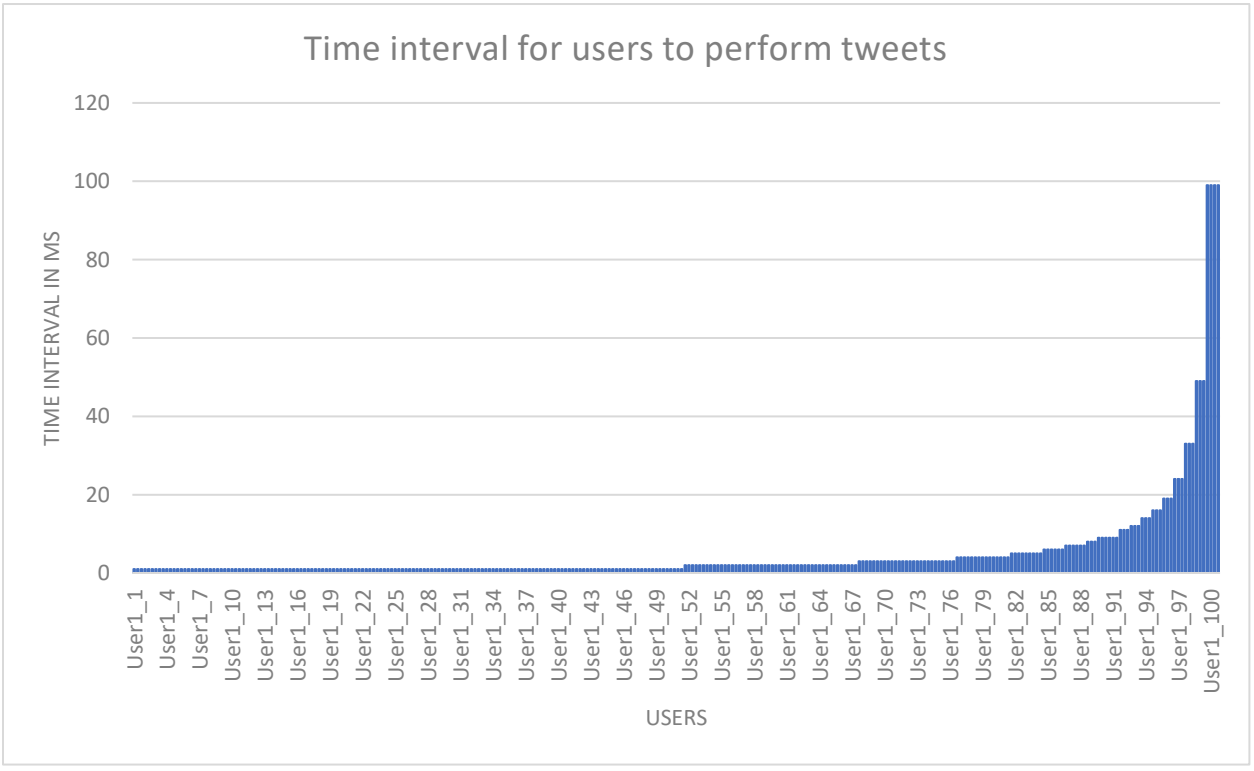
ii. Time intervals for user to perform tweets for each user ranging from 1 to 100 (**With 1 client and 100 users**):



iii. Zipf distribution on number of subscribers for each user ranging from 1 to 100 (**With 3 clients and 300 users**):



iv. Time intervals for user to perform tweets for each user ranging from 1 to 100 (**With 3 clients and 300 users**):



10. Performance Evaluation:

i. Performance evaluation varying number of users for 100,000 requests and with 1 client:

Average time for each service in ms				
No of Users	**Tweets**	**Retweets**	**Query Hashtags**	**Query Mentions**
100	16.411	17.714	16.387	17.382
500	20.589	21.245	22.058	21.318
1000	19.295	19.600	23.013	17.763
2000	141.582	184.211	164.365	163.965
5000	533.265	233.345	612.324	563.232
8000	893.876	212.653	845.231	414.242

ii. Performance evaluation varying number of users for 100,000 requests and with 3 client:

Average time for each service in ms				
No of Users	**Tweets**	**Retweets**	**Query Hashtags**	**Query Mentions**
100	50.874	56.6848	49.161	48.669
500	265.884	270.108	277.203	263.954
1000	620.708	686.600	667.133	666.113
2000	893.423	532.231	782.342	712.523
5000	1433.432	982.633	1022.342	1345.643
8000	2031.424	1695.634	1425.643	1564.745

11. Largest number of users simulated:

We have simulated a total of 50000 users with three clients, client 1 simulating 15000 users, client 2 simulating 20000 users and client 3 simulating 15000 users.