COP5615: Distributed Systems

Project 4 - Part I – Tweeter Report

Group Members:

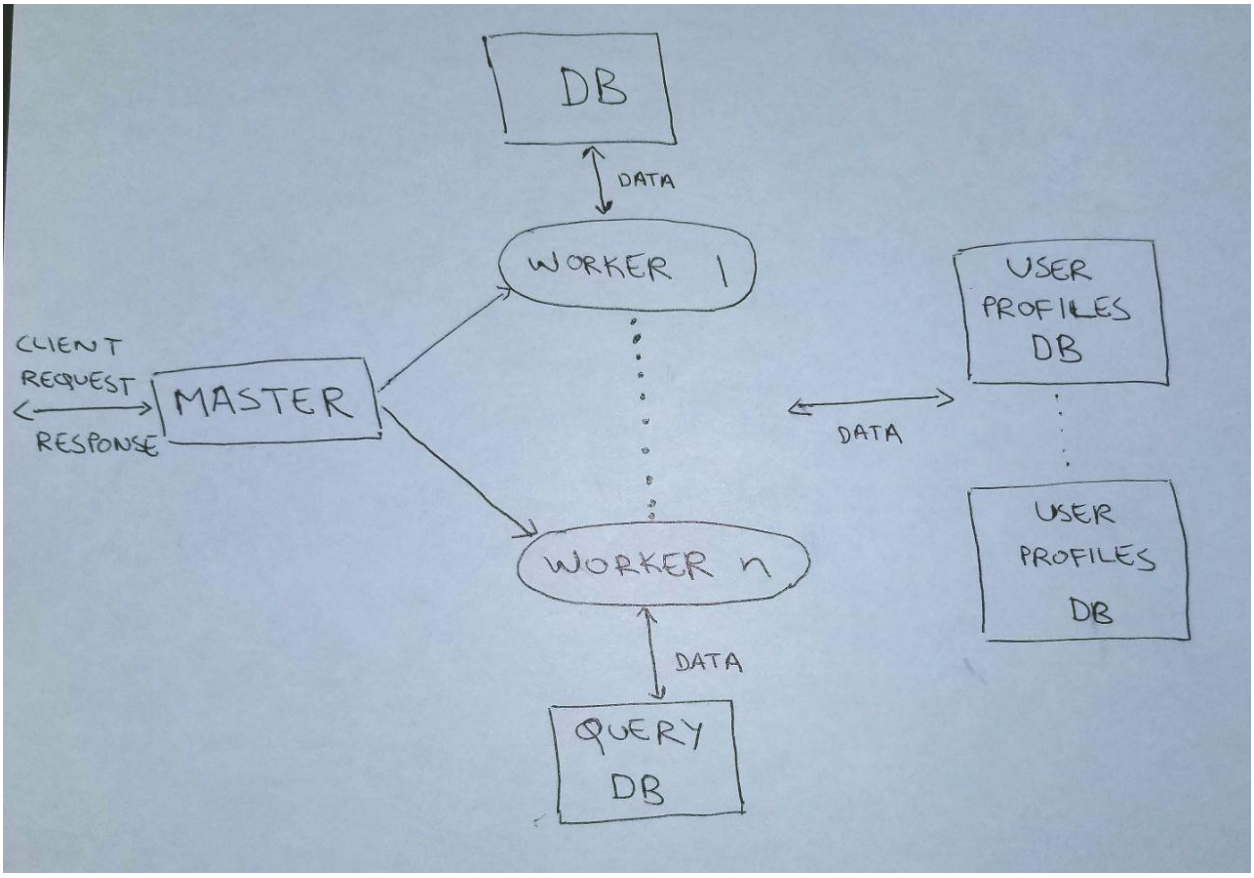
Raghav Ravishankar, UFID – 19995874

Suhas Kumar Bharadwaj, UFID – 16120229

Objective:

To implement a Twitter clone with a simulator client and the engine (server).

Engine Architecture:



Engine Architecture

On starting the engine, a node is created. This node then calls upon the master which is a GenServer process. The master handles all work distribution in the engine and is the resource manager. It also acts as the interface between the client and the engine. The master on starting initializes the following GenServer processes:

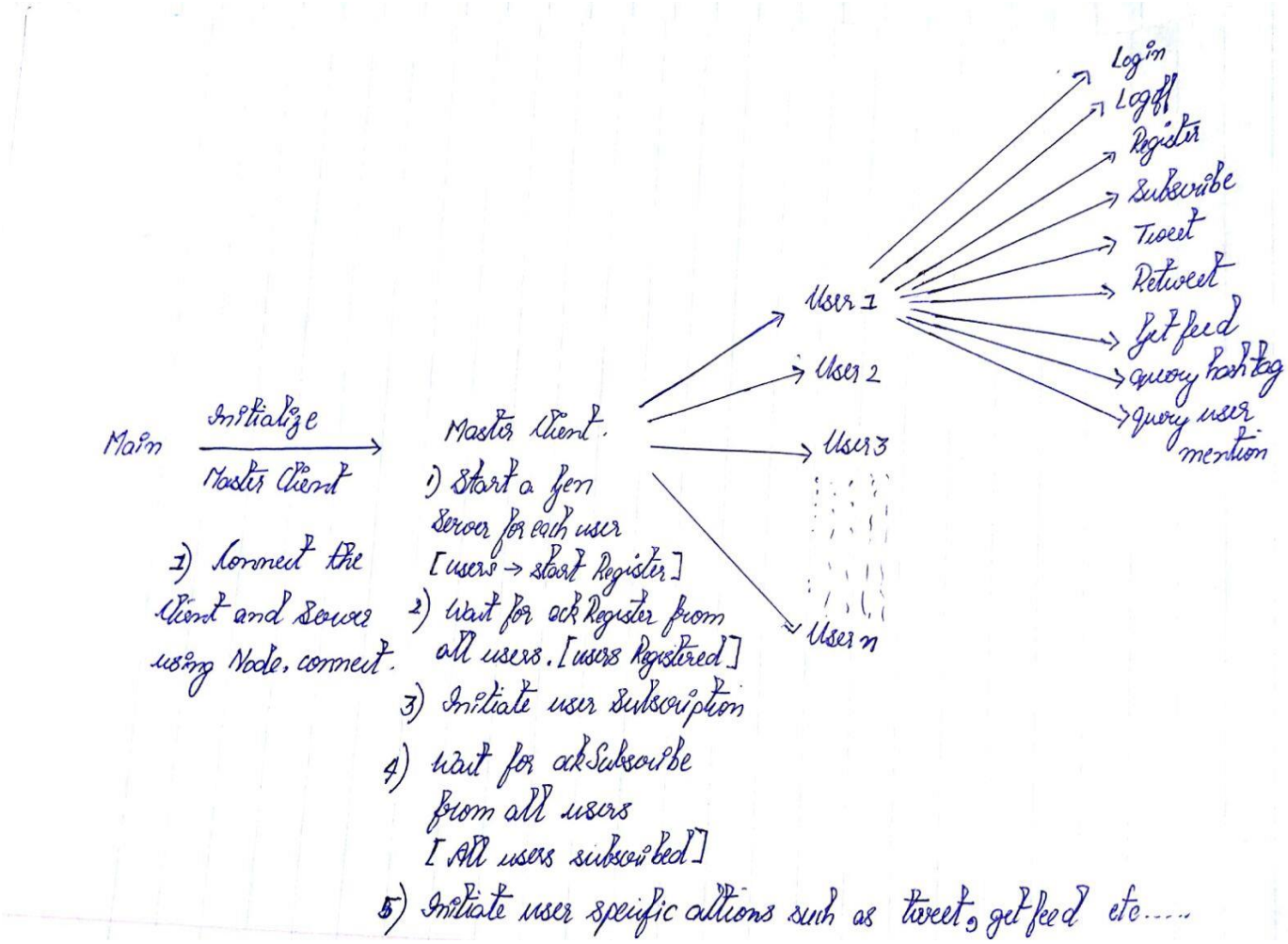
1. DB: One GenServer process is started to act as the common tweets database for the engine. The DB holds the following data:
    i. A map called **tweets** contains a mapping between a worker created tweet_id and the tweet sent by the client. This map holds all the tweets that is passed to the engine. The tweet_id is generated by the worker. It is a concatenation of the workers unique tag(name) and the current count of tweets that the worker has handled.
    ii. A map called **userID_to_userpDB_num_map**. The master creates one/more user profiles database that stores the data of a set of users. Every time a new user registers he is allotted their dataspace in one of the user profile DBs. The users' data is always stored in that corresponding user profile DB. Hence the said map holds the mapping of each users' user_id and which user profile DB their data is stored in.
    iii. The last data structure is the **all_users_map** which contains the map between all the valid user_ids and their password. This is used every time an old user logs in for authentication.
2. Query DB: This also is just one GenServer process. This acts like another database but is used to support some of the functionality that the engine performs. The different data structures in this DB are:
    i. A map called **hashTags**, which stores all the hash-tags encountered by the engine and maps it to a list of tweet_ids that contained the hash-tag. This helps in the functionality of returning all the tweets that contain a queried hash-tag.
    ii. The **mentioned** map is similar to the hashTags map. It stores the user_id and maps it to a list of all tweets that mentioned the valid user_id. This helps when a user queries to see all the tweets that mentions a particular user_id.
    iii. The **activePIDs** maps the user_ids that are currently logged-in into the application and their PIDs. This is used to send live feeds to the active users.
3. User Profiles DB: The master starts one or more GenServer processes that are used to hold the user profile data. This process contains a single map, the userData map. This map has the user_id as its key. The value is a list that contains three other lists with respect to the user. The first list contains the tweet_ids of all the tweets sent by the user. The second list contains the tweet_ids of all the tweets that the user has subscribed to. The third list contains the user_ids of all the users that have subscribed to this particular user.
4. Worker: The master starts one or more GenServer processes that act as workers. The worker performs all the necessary tasks to ensure the completion of each functionality of the engine. It takes in work sent to the master from the clients and executes this by querying the different database processes necessary to complete the task. Each worker has a counter of the number of new tweets that they have handled. It also has a specific worker tag

associated to it like “w1”, “w2”, etc. It also contains the number of user profile database processes that have been created by the master.

After setting up this architecture the master waits for requests from the client side. The request is processed by forwarding it to one of the workers that have been created. The worker is picked randomly each time to ensure good distribution of work between the workers in the engine. If the request from the client has an appropriate response, then it is relayed back to the client by the master. Two levels of authentication are performed by the engine. The first level is by associating a user_id and password to each user of the system. Each time the user logs in, these credentials are validated. The second level is ensured by providing a session_id each time a user logs in. This id must be used for every subsequent request by the user until they log off. A new session_id is provided the next time the user logs in.

The engine doesn’t use a database. It uses processes as data stores. One constraint of the engine is that all the tweets are stored in a centralized process called DB. The DB might be overloaded when there are many users. Some constraints are imposed on the user. The first one is that the user-id and password must be a string. The second one is that the user is logged-in when they register and hence they can send the next log-in call only after they log off. The number of user profile database processes and workers that is created by the master can be tuned programmatically.

Client Architecture:



1. Master Client: It starts a Gen Server for all the user clients. It initiates the registration and subscription for all the users and keeps a track of the registered clients and subscribed clients. Once all the users have finished subscribing, it initiates the user specific actions such as tweeting, query hashtag, query user mention etc. in a random manner.

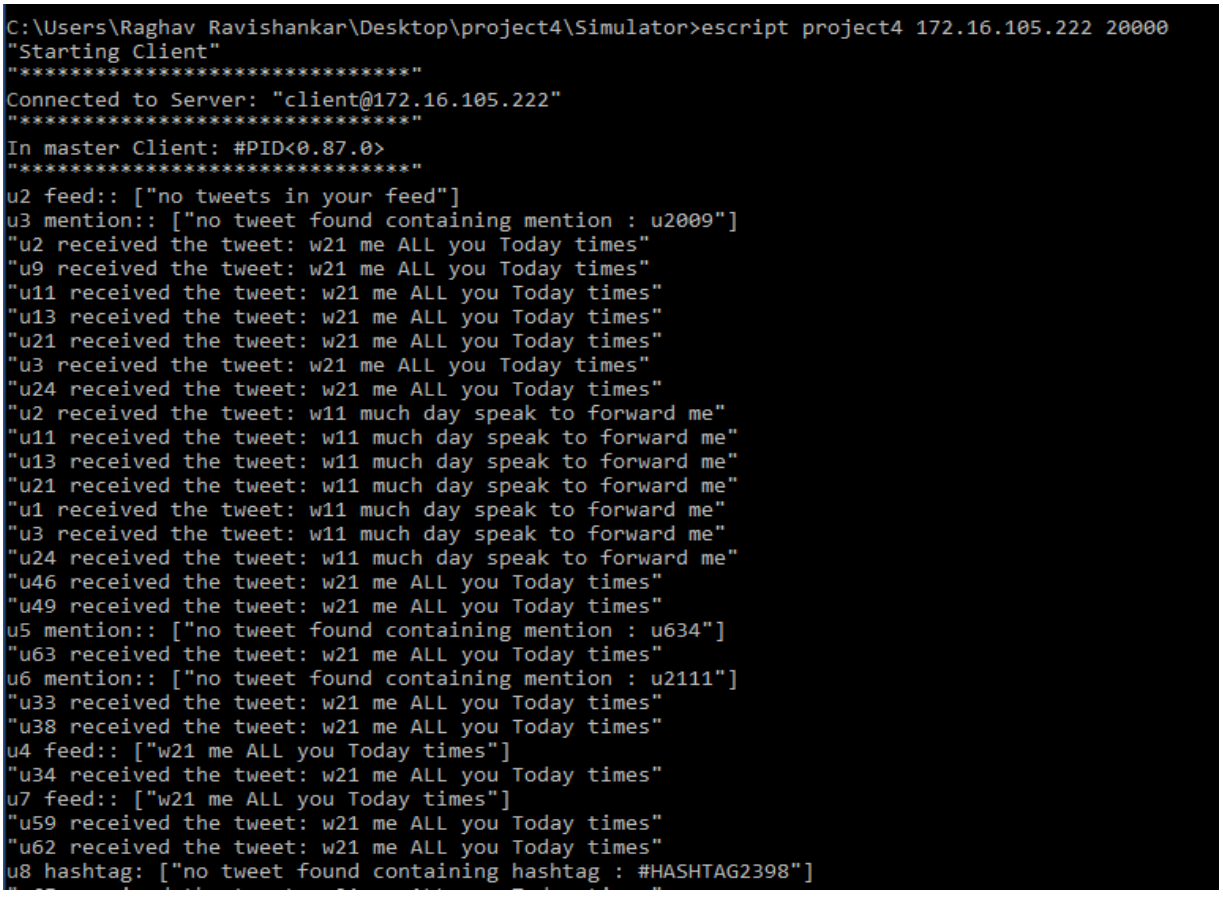
2. User Client: Each user client randomly generates a tweet consisting of random hashtags and user mentions according to the zipf distribution i.e. number of followers is proportional to the frequency of tweets. Once the tweets are generated and ready for use, it randomly performs operation such as tweeting, querying hashtags, querying user mentions, getting it's feed etc.

Functionality:

The tweeter engine can perform all the necessary functions requested. A new user can register with a user-id and password. The user can subsequently login and logoff from the application using the required session-id. A user can subscribe to any other existing user through their user-id. The logged-in user can send out tweets. These tweets are displayed to all the active users who have subscribed to this particular user. A user can re-tweet tweets from other users he is subscribed to. Every time a user logs-in, the 5 most recent tweets in his feed are displayed. A user can get the top 5 most recent tweets in his feed by querying the application. The user can also query the engine based on a particular hashtag or mention. This returns all the tweets containing the queried request.

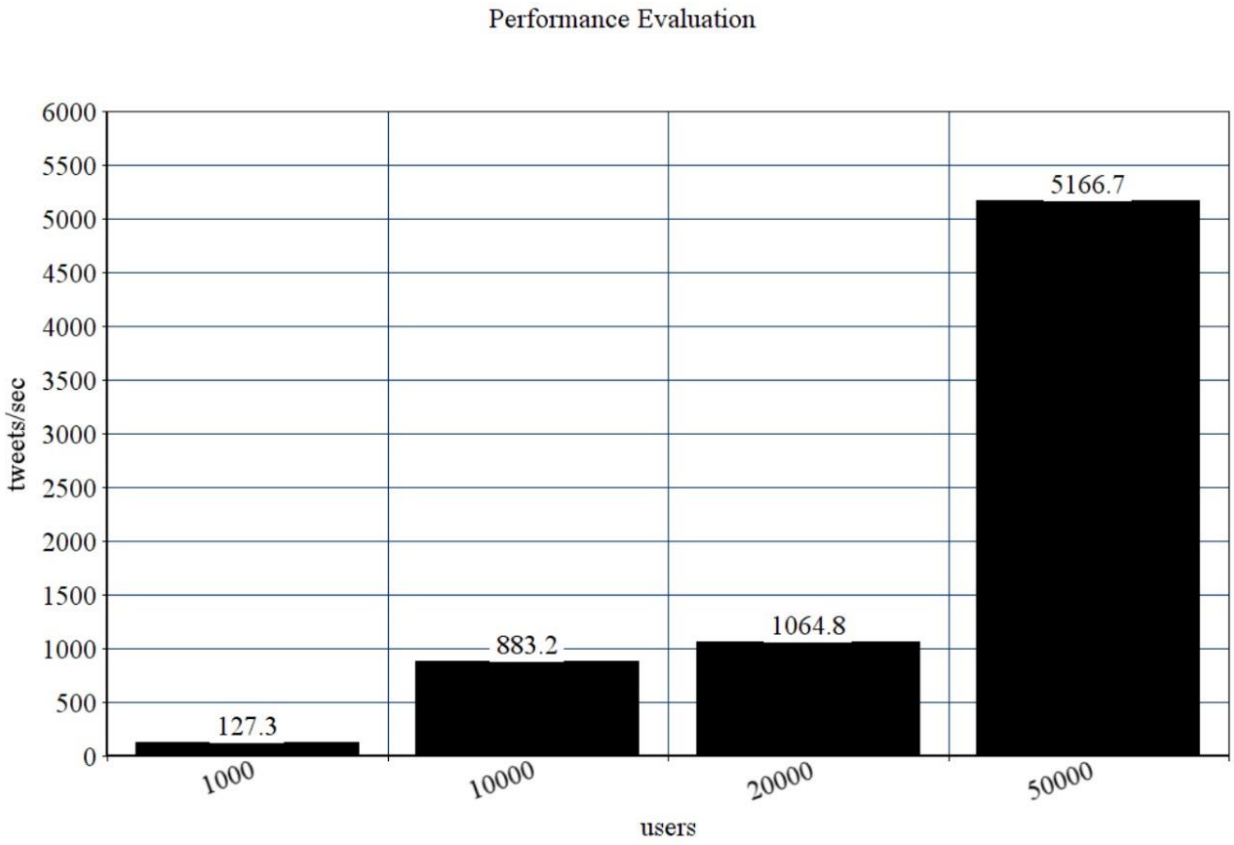
Testing:

The testing involves the client-side users generating tweets according to the zipf distribution i.e. more the number of followers, the frequency of tweeting will also be equally more. While the tweeting is happening in the background, an operation (tweet, login, logoff, query hashtag etc.) is chosen randomly and executed.



An example screenshot of executing the tweeter engine for 20,000 users

Performance:



The Engine was tested for varying number of users. The performance metric used was the number of tweets being handled by the engine per second for these users. The performance was taken for 5 five-second windows and averaged out. The results are shown in the graph above. The tweets were counted as they were entered into the tweets map in DB.

How to run:

1. project4 is the .zip folder containing both Engine and Simulator.

2. Extract the contents of the project4.zip folder into a folder named project4.

2. Build engine:

    a. cd ~/project4/Engine

    b. mix escript.build

3. Execute Engine:

    a. escript project4 - prints the IP address of the machine running this application

    Example:

```
C:\Users\Raghav Ravishankar\Desktop\EngineF\project4>escript project4
"Master Node created"
"server@172.16.105.222"
"Master Server started"
```

4. Build client:

    a. cd ~/project4/Simulator

    b. mix escript.build

5. Execute Client:

    a. escript project4 serverIP No. of Clients

    Example:

```
C:\Users\Raghav Ravishankar\Desktop\project4\Simulator>escript project4 172.16.105.222 10
"Starting Client"
"********************************"
Connected to Server: "client@172.16.105.222"
"********************************"
In master Client: #PID<0.87.0>
"********************************"
```