COP5612 – Fall 2020 / Project 4 Part 1

TEAM MEMBERS

— Hsiang-Yuan Liao, UFID:4353-5341, hs.liao@ufl.edu
— Tung-Lin Chiang, UFID:9616-8929, t.chiang@ufl.edu

FILES

```
Twitter_Engine
  ├ Collections.fs
  ├ program.fs
  └ directory file for F# project (.fsproj)

Twitter_Simulator
  ├ program.fs
  └ directory file for F# project (.fsproj)
```

GOAL

The goal of this project is to implement a Twitter engine and a simulator. The engine provides functionalities, such as registration, Tweet, Retweet, subscription, and querying tweets. The simulator can control the numbers of users, connections of users and make the subscriptions number Zipf-distributed.

HOW TO RUN THE PROGRAMS

Twitter Engine:

One can simply use the command below to start the engine.

➡ ***dotnet run***

Twitter Simulator

There are three modes in our simulator : (A) **SIMULATION** (B) **DEBUG** (C) **USER**

To run the program in **SIMULATION** mode:

➡ **dotnet run simulate**

To run the program in **DEBUG** mode:

➡ **dotnet run debug**

To run the program in **USER** mode:

➡ **dotnet run user**

(A) **SIMULATION MODE:**

One can not only run the program in our designed simulation scenario, but can also setup the parameters below:

- ***numClients***:

  Number of users
- ***percentActive***:

  Percentage of active users
- ***percentSendTweet***:

  Percentage of active users who send a tweet
- ***percentRetweet***:

  Percentage of active users who Retweet
- ***percentQueryHistory***:

  Percentage of active users who query previous tweets of a random user
- ***percentQueryByTag***:

  Percentage of active users who query tweets having a mention of a random user
- ***percentQueryByMention***:

  Percentage of active users who query tweets having a random hashtag
- ***totalRequest:***

  Maximum of total requests sent to the server
- ***maxCycle:***

  Maximum of times to repeat the simulation

The detail about our designed simulation scenario is written in the **DEBUG** mode.

(B) **DEBUG MODE:**

one can run the program in our designed simulation scenario below.

In our designed scenario, there are two parts: ***basic setup***, and ***repeated simulations***.

BASIC SETUP

1. Spawn:

Spawn ***numClients*** client actors

2. Registration & Connect:

Register an account for each clients and connect to the server.

3. Subscription:

Base on the Zipf distribution and ***numClients***, randomly assign the needed number of subscriptions for each client. Then, randomly select clients to subscribe and make each client reach the assigned number of subscriptions.

4. Tweet:

Each client is assigned a random number ***T = t*** * (# of subscriptions), where

1 ≤ ***t*** ≤ 5, and has to send ***T*** tweets. Each tweet has a randomly hashtag and mentions a randomly selected client. Note that there are a set of default hashtags in our simulator.

5. Disconnect:

Make all clients disconnected.

REPEATED SIMULATIONS

Our simulator repeat the steps in every 0.5 seconds.

1. Randomly select ***numInActive*** connected clients to disconnect

2. Randomly select ***numActive*** disconnected clients to connect.

3. Randomly select ***numSendTweet*** connected clients send tweets with a random hashtag and a random client to mention.

4. Randomly select ***numRetweet*** connected clients to Retweet a random tweet.

5. Randomly select ***numQueryHistory*** connected clients to query all tweets of a random client.

6. Randomly select ***numQueryByTag*** connected clients to query all tweets having a random tag.

7. Randomly select ***numQueryByMention*** connected clients to query all tweets which mentioned a random client.

To stop the stage of repeated simulations, we have two variables to control our simulator, ***maxCycle*** and ***totalRequest***. The stage of repeated simulations stops when the number of repeated times reach ***maxCycle*** or the server has received ***totalRequest*** requests.

Here are our default values in debug mode:

- *maxCycle = 1000*
- *numClients = 1000*
- *totalRequest = 2147483647*
- *percentActive = 30 (%)*
  *numInActive = (# of active clients) * (percentActive - 1)*
  *numActive = (# of inactive clients) * percentActive*
- *percentSendTweet = 50 (%)*
  *numSendTweet = numActive * percentSendTweet * 100%*
- *percentRetweet = 20 (%)*
  *numRetweet = numActive * percentRetweet * 100%*
- *percentQueryHistory = 20 (%)*
  *numQueryHistory = numActive * percentQueryHistory * 100%*

- *percentQueryByTag = 10 (%)*
  *numQueryByTag = numActive * percentQueryByTag * 100%*
- *percentQueryByMention = 10 (%)*
  *numQueryByMention= numActive * percentQueryByMention * 100%*

Note that the clients are different in the three sets of clients of ***numQueryHistory***, clients of ***numQueryByTag***, and clients of ***numQueryByMention***

(C) **USER MODE**:

One can run the program as a twitter user to communicate with the server.

In the beginning, you can

(1) Register your own account
(2) Connect to the server to login if there is a match between the provided information and an account in server.

When you have logged in, you can

(1) Post a Tweet

(2) Retweet a Tweet

(3) Subscribe to a user

(4) Disconnect / Log out

(5) Query a User's History Tweets

(6) Query Tweets with a hashtag

(7) Query Tweets for a mentioned User

(8) Query subscribe status for a User

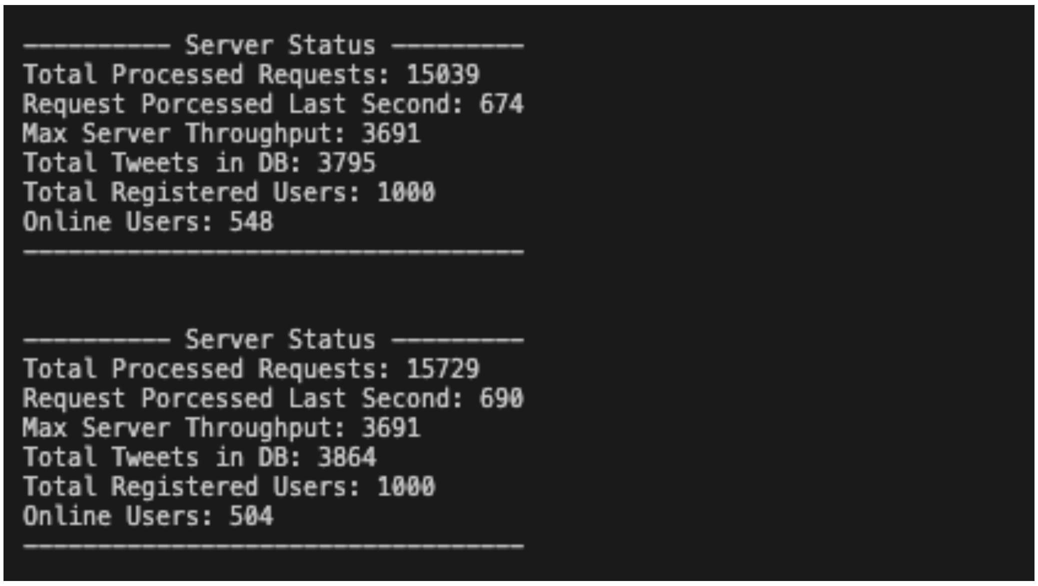
(9) Terminate this program

OUTPUTS

Twitter Engine

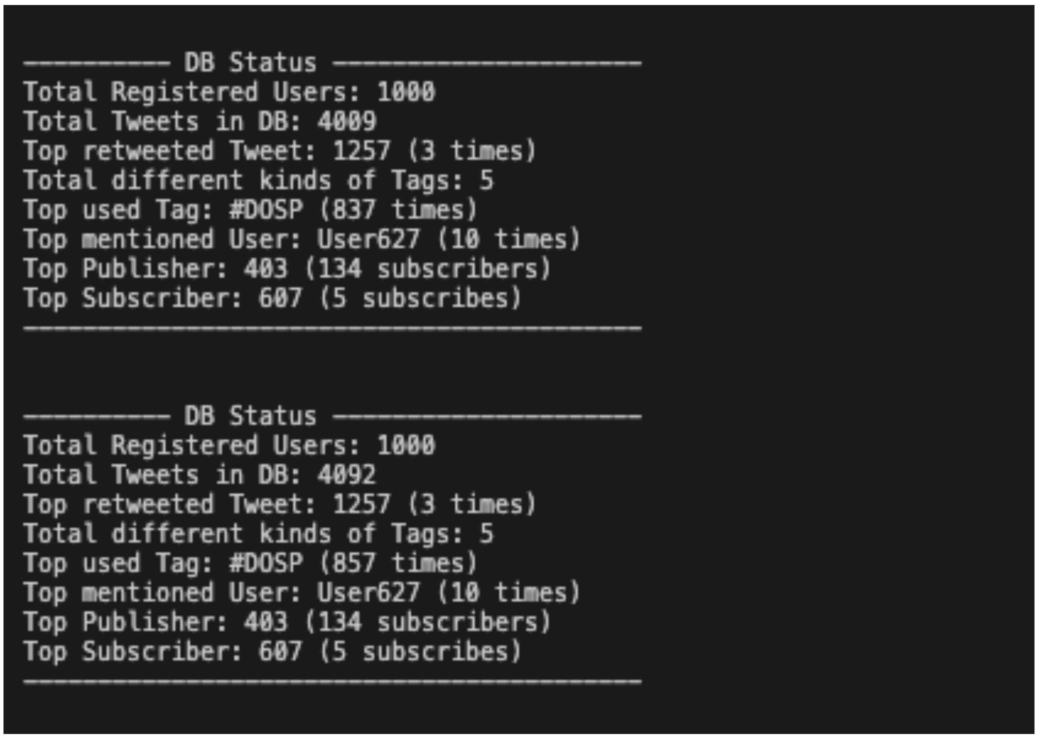
There are two kinds of output can be printed every one second. One can press any key to switch between the output statuses.

1. Server Status



```
---------- Server Status ---------
Total Processed Requests: 15039
Request Porcessed Last Second: 674
Max Server Throughput: 3691
Total Tweets in DB: 3795
Total Registered Users: 1000
Online Users: 548
----------------------------------


---------- Server Status ---------
Total Processed Requests: 15729
Request Porcessed Last Second: 690
Max Server Throughput: 3691
Total Tweets in DB: 3864
Total Registered Users: 1000
Online Users: 504
----------------------------------
```

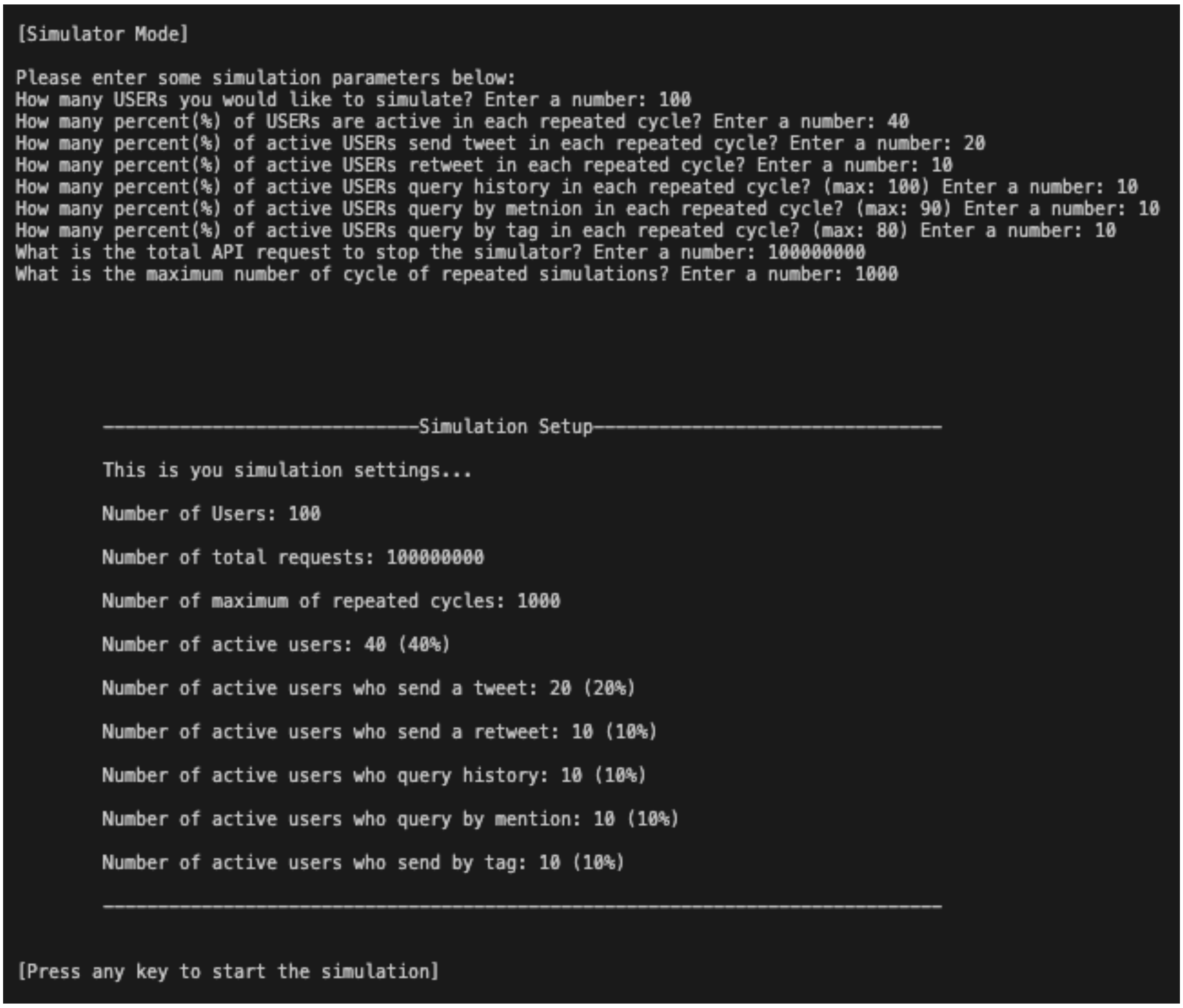
2. DB Status



```
---------- DB Status ---------------------
Total Registered Users: 1000
Total Tweets in DB: 4009
Top retweeted Tweet: 1257 (3 times)
Total different kinds of Tags: 5
Top used Tag: #DOSP (837 times)
Top mentioned User: User627 (10 times)
Top Publisher: 403 (134 subscribers)
Top Subscriber: 607 (5 subscribes)
------------------------------------------


---------- DB Status ---------------------
Total Registered Users: 1000
Total Tweets in DB: 4092
Top retweeted Tweet: 1257 (3 times)
Total different kinds of Tags: 5
Top used Tag: #DOSP (857 times)
Top mentioned User: User627 (10 times)
Top Publisher: 403 (134 subscribers)
Top Subscriber: 607 (5 subscribes)
------------------------------------------
```

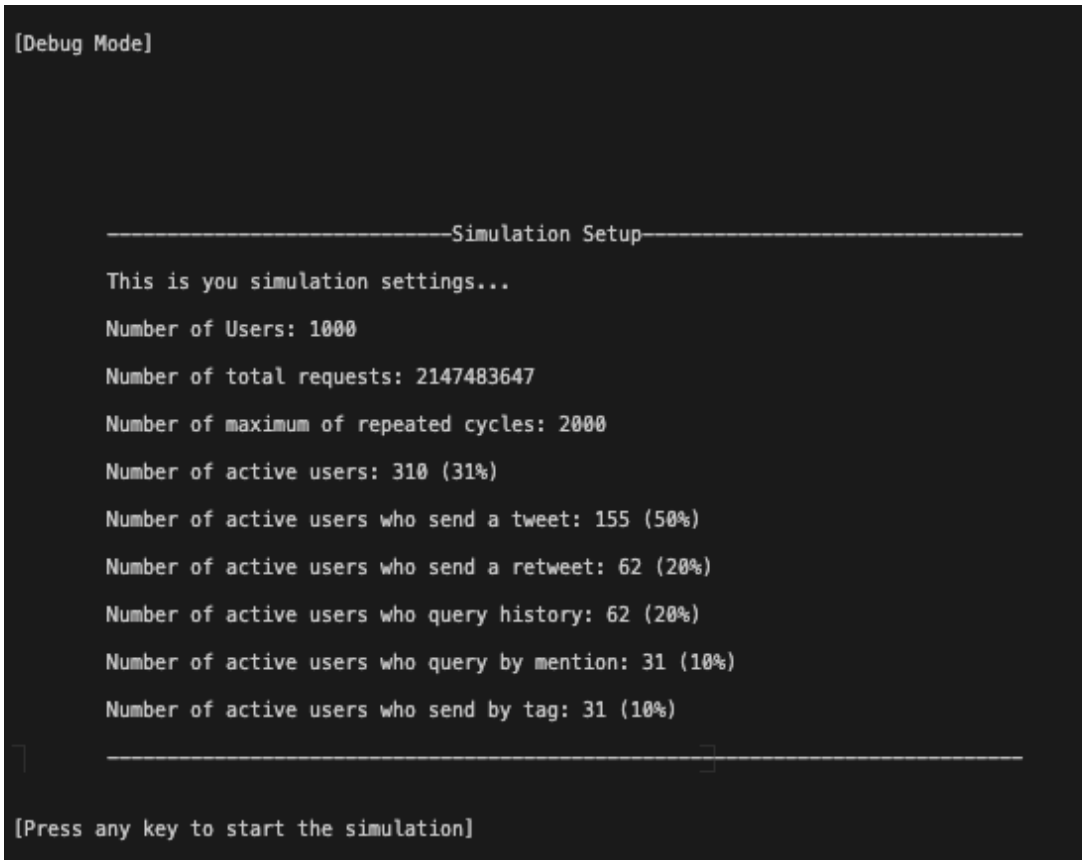
Twitter Simulator

1. **SIMULATION MODE:**



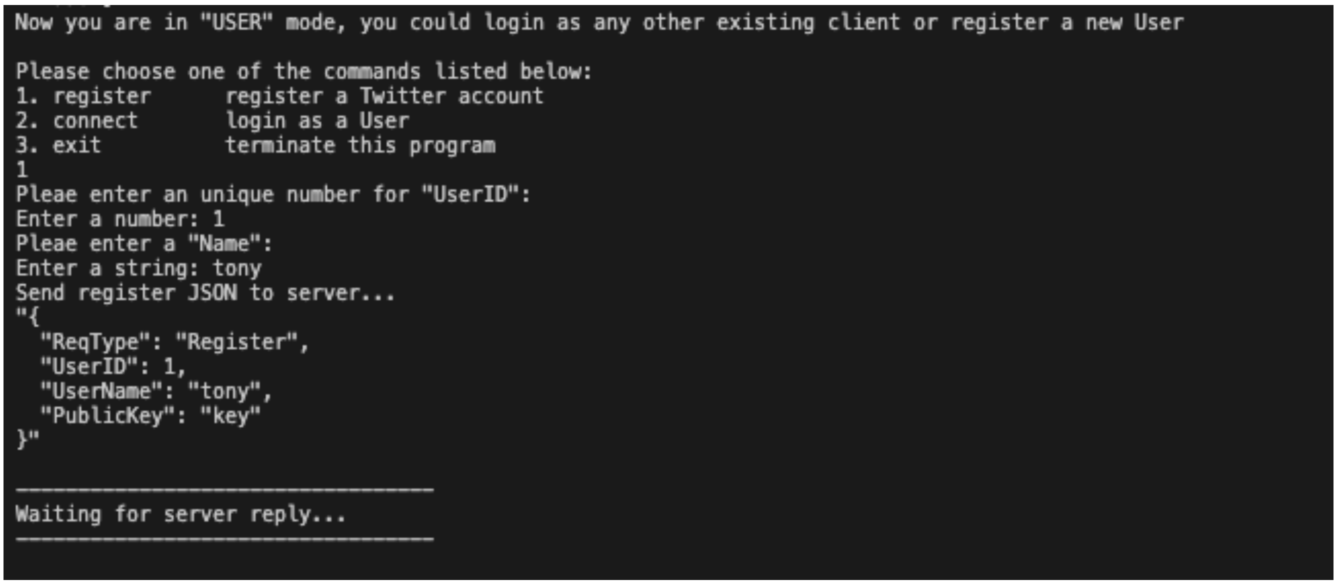
```
[Simulator Mode]

Please enter some simulation parameters below:
How many USERs you would like to simulate? Enter a number: 100
How many percent(%) of USERs are active in each repeated cycle? Enter a number: 40
How many percent(%) of active USERs send tweet in each repeated cycle? Enter a number: 20
How many percent(%) of active USERs retweet in each repeated cycle? Enter a number: 10
How many percent(%) of active USERs query history in each repeated cycle? (max: 100) Enter a number: 10
How many percent(%) of active USERs query by metnion in each repeated cycle? (max: 90) Enter a number: 10
How many percent(%) of active USERs query by tag in each repeated cycle? (max: 80) Enter a number: 10
What is the total API request to stop the simulator? Enter a number: 100000000
What is the maximum number of cycle of repeated simulations? Enter a number: 1000




        -----------------------------Simulation Setup--------------------------------

        This is you simulation settings...

        Number of Users: 100

        Number of total requests: 100000000

        Number of maximum of repeated cycles: 1000

        Number of active users: 40 (40%)

        Number of active users who send a tweet: 20 (20%)

        Number of active users who send a retweet: 10 (10%)

        Number of active users who query history: 10 (10%)

        Number of active users who query by mention: 10 (10%)

        Number of active users who send by tag: 10 (10%)

        -----------------------------------------------------------------------------


[Press any key to start the simulation]
```

2. **DEBUG MODE:**

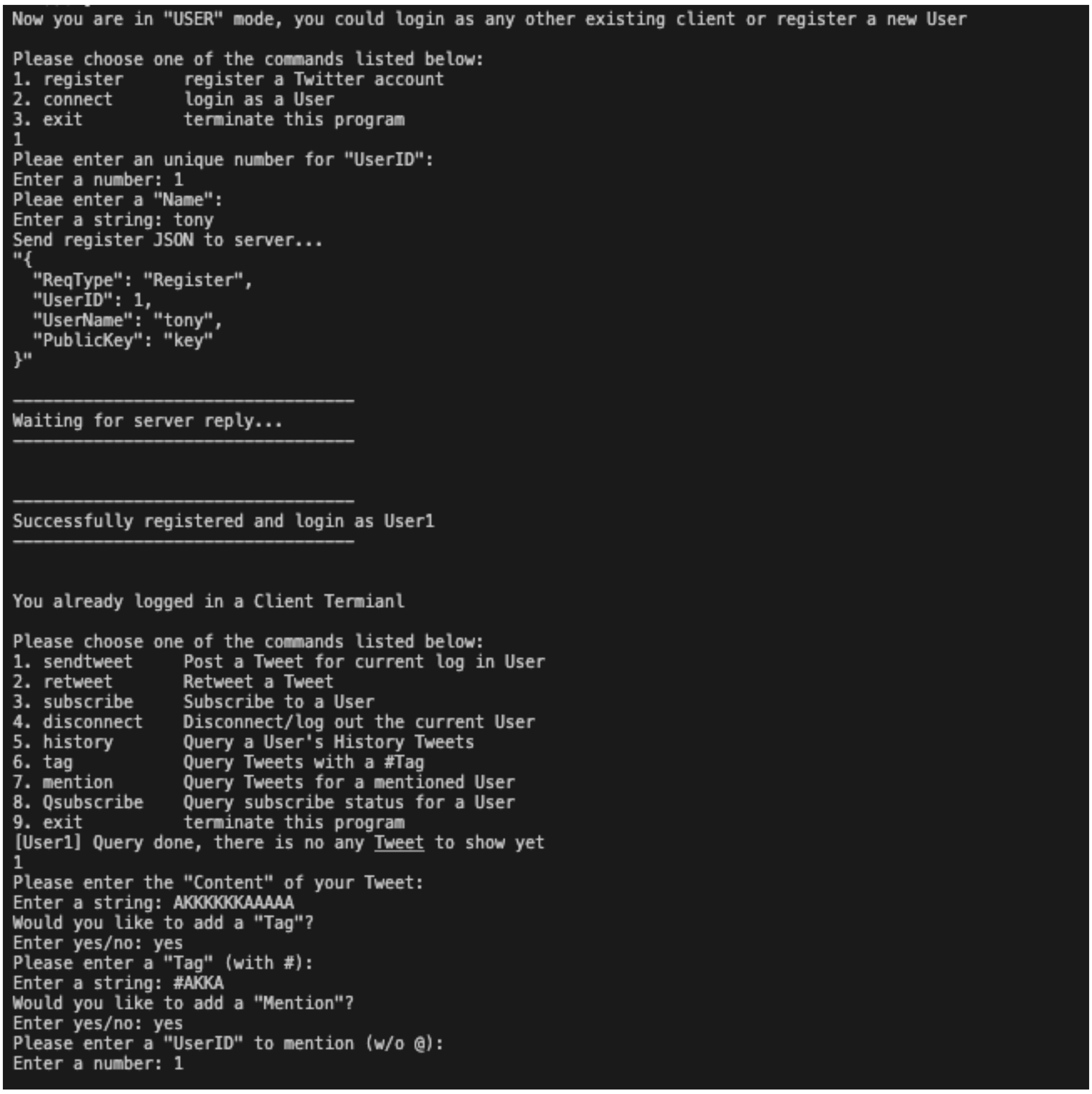


```
[Debug Mode]




        ------------------------------Simulation Setup---------------------------------

        This is you simulation settings...

        Number of Users: 1000

        Number of total requests: 2147483647

        Number of maximum of repeated cycles: 2000

        Number of active users: 310 (31%)

        Number of active users who send a tweet: 155 (50%)

        Number of active users who send a retweet: 62 (20%)

        Number of active users who query history: 62 (20%)

        Number of active users who query by mention: 31 (10%)

        Number of active users who send by tag: 31 (10%)

        ------------------------------------------------------------------------------


[Press any key to start the simulation]
```

3. **USER MODE:**



```
Now you are in "USER" mode, you could login as any other existing client or register a new User

Please choose one of the commands listed below:
1. register      register a Twitter account
2. connect       login as a User
3. exit          terminate this program
1
Pleae enter an unique number for "UserID":
Enter a number: 1
Pleae enter a "Name":
Enter a string: tony
Send register JSON to server...
"{
  "ReqType": "Register",
  "UserID": 1,
  "UserName": "tony",
  "PublicKey": "key"
}"

-----------------------------------
Waiting for server reply...
-----------------------------------
```



```
Now you are in "USER" mode, you could login as any other existing client or register a new User

Please choose one of the commands listed below:
1. register      register a Twitter account
2. connect       login as a User
3. exit          terminate this program
1
Pleae enter an unique number for "UserID":
Enter a number: 1
Pleae enter a "Name":
Enter a string: tony
Send register JSON to server...
"{
  "ReqType": "Register",
  "UserID": 1,
  "UserName": "tony",
  "PublicKey": "key"
}"

-----------------------------------
Waiting for server reply...
-----------------------------------



-----------------------------------
Successfully registered and login as User1
-----------------------------------



You already logged in a Client Termianl

Please choose one of the commands listed below:
1. sendtweet     Post a Tweet for current log in User
2. retweet       Retweet a Tweet
3. subscribe     Subscribe to a User
4. disconnect    Disconnect/log out the current User
5. history       Query a User's History Tweets
6. tag           Query Tweets with a #Tag
7. mention       Query Tweets for a mentioned User
8. Qsubscribe    Query subscribe status for a User
9. exit          terminate this program
[User1] Query done, there is no any Tweet to show yet
1
Please enter the "Content" of your Tweet:
Enter a string: AKKKKKKAAAAA
Would you like to add a "Tag"?
Enter yes/no: yes
Please enter a "Tag" (with #):
Enter a string: #AKKA
Would you like to add a "Mention"?
Enter yes/no: yes
Please enter a "UserID" to mention (w/o @):
Enter a number: 1
```

IMPLEMENTATION

In our Twitter Engine and Simulator, all the actors communicate on JSON messages. It pretty provides a simple way to communicate between different processes. Since JSON messages are the sting type. All we need is to let the actors to expect string type message. It is like a common language in our system.

On the sever side, we have a master server to receive all the requests from the client side. To have a better performance, we also use multiple (1000) actors named queryWorkers to help the master server deal with those time-consuming query requests such as queries of all previous tweets. When the master server received a query request, it would randomly assign this request to one of the queryWorkers, and the client can just wait for the reply from the worker.

On the client side, we make a restriction for the clients. That is, when a client is waiting for the reply from its last request, the client would discard any other orders from the simulator and keep waiting until receiving the reply

MEASUREMENT & RESULT

In order to collect data, we we make the server keep printing its status every one second.

To measure the performance, we try different number of clients to in SIMULATION mode. When the simulation is in the stage of repeated simulations, there are a great number of requests sent to the server. We record the data from the server status to calculate the **averaged processed request per second**:

# of clients	Averaged processed request per second
500	742
1000	771
3000	1076
5000	1873

Also, in the stage of repeated simulations, we run another process in USER mode and record the **response time (seconds)** for different request.

# of clients	Disconnect	Tweet	ReTweet	Subscribe	Query history	Query by tag	Query by mention
1000	0.02	0.06	0.03	0.041	11.89	4.23	5.59
3000	5.76	8.41	5.62	12.77	31.62	15.21	7.36
5000	6.82	6.58	14.78	16.31	15.87	14.25	8.58

Therefore, we can see that when the traffic on the server gets heavy, the response time for each requests increases. It is interesting that when there are less data to send as the reply for the query requests, the response time could be lower than expected. Also, we found that in the beginning of the stage of repeated simulations, the response time for each request are smaller than the records above. Based on this observation, we think that the reason of this issue could be that the size of the request queue of the master server's mailbox is not too large.

Difficulties and Issues and future

1.Massive Query Requests:

Query requests costs a lots of server resource since the server has to find out all the target Tweets that match the query condition in database-like collections. For example, if a user query a specific "Tag" that is used in N Tweets. Then the server has to find out all the N Tweets and then send them back to the user. When there are more and more Tweets stores in the server side, each query request is more likely to decrease the server throughput.

To fix this problem, we create tons of Query-Worker-Actors to help our server dealing with query requests. Once our server received a query request, the server will check if this is a valid request. If the request is validated, the server will forward the query request to a randomly picked query-worker-actor out of the worker pool. The query worker will do the rest of the work for server, which gives the opportunity for server to process other in coming requests.

2. Message Passing:

At the beginning of this project, we faced some difficulties to communicate between different processes if we don't have a uniform message type. For the aspect of maintenance and future compatibility, we decide to use JSON as our uniform message format and type.

3. Database Concurrency:

It is also a big issue to keep database concurrency and to avoid concurrent read-write to database. In our Twitter Engine, the data storage is temporarily implemented by F# generic collections which is stored in program memory. Because they are shared memory in our Sever we have to avoid concurrent read-write issues.

For now, the server is the only actor that assigning write tasks to a specific collection. Although it prevents the concurrent read-write risk, it truly increases the loading for server. The server cannot process any other requests until a specific write task caused by the current request is done. It is also a downside for server throughput.

To deal with this concurrent read-write issue, we have two thoughts to try in the project part 2. First, we may add more worker actors for each generic collection. When ever a read-write task is needed. We could let the server simply forward write requests to that actor. And the server will only need to validate the requests. Second, we would like to implement a SQLite database into the server. It might have a great performance dealing with massive datas and meanwhile take good care of concurrent read-write issue.