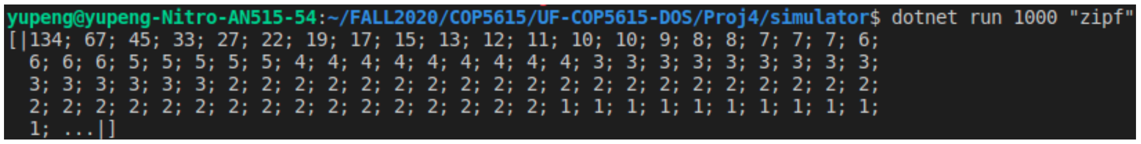
COP 5615 – FALL 2020
Project 4 Part I – Twitter Clone

Yu-Peng Chen
7904-3193

1. Instructions
   Two independent processes are implemented. One is the server (engine) process, the other is the client process. For the client process, I implemented a simulator to activate multiple clients and simulate the clients' behaviors. The instruction of running the program as follows,
   A. Testing phase
   The goal of this phase is to test the correctness of the server APIs. Thus, **there is an option of printing the debugging information to the pre-existing folder "log"**, which is placed under the same directory as the "Client.fs" file. The logging option will print out a .txt file with the name of "log[ID].txt" for each client, so it is recommended that this option should be run with a small number of clients, otherwise there will be a lot of text files written to the "log" folder. The following is the command line for this phase:
   - First, navigate to the "**server**" folder and run the following commands:
     - dotnet build
     - dotnet run

     This we keep the process running and listening to requests.
   - Then, navigate to the "**simulator**" folder and run the following commands:
     - dotnet build
     - dotnet run [NUMBER_OF_USER] [MODE] [DEBUG]
       - Example: dotnet run 50 "testing" "true"
       - NUMBER_OF_USER: An integer indicating the number of simulated users.
       - MODE: A string that should be set to **"testing"** in this phase.
       - DEBUG: A string that can be set to "true" or "false".

   B. Zipf phase
   The goal of this phase is to simulate the Zipf distribution ($X \sim Zipf(\alpha, n)$) with the probability mass function: $f(x) = \frac{1}{x^{\alpha} \sum_{i=1}^{n} (\frac{1}{i})^{\alpha}}, where\ x = 1, \dots n$. In the Zipf simulation for this project, n is the number of total registered clients, and the distribution provides **the number of subscribers** for each client. Here is an example when n = 1000 and $\alpha = 1$:



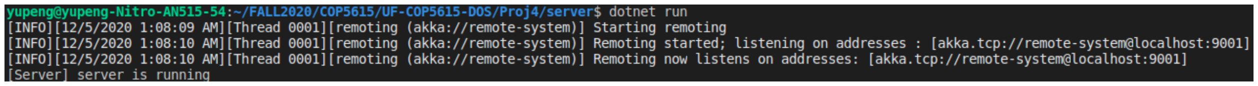
```
yupeng@yupeng-Nitro-AN515-54:~/FALL2020/COP5615/UF-COP5615-DOS/Proj4/simulator$ dotnet run 1000 "zipf"
[|134; 67; 45; 33; 27; 22; 19; 17; 15; 13; 12; 11; 10; 10; 9; 8; 8; 7; 7; 7; 6;
  6; 6; 6; 5; 5; 5; 5; 5; 4; 4; 4; 4; 4; 4; 4; 4; 4; 3; 3; 3; 3; 3; 3; 3; 3; 3;
  3; 3; 3; 3; 3; 3; 2; 2; 2; 2; 2; 2; 2; 2; 2; 2; 2; 2; 2; 2; 2; 2; 2; 2; 2; 2;
  2; 2; 2; 2; 2; 2; 2; 2; 2; 2; 2; 2; 2; 2; 2; 2; 1; 1; 1; 1; 1; 1; 1; 1; 1; 1;
  1; ...|]
```

   - First, navigate to the "**server**" folder and run the following commands:
     - dotnet build
     - dotnet run

     This we keep the process running and listening to request.
   - Then, navigate to the "**simulator**" folder and run the following commands:
     - dotnet build
     - dotnet run [NUMBER_OF_USER] [MODE] [DEBUG]
       - Example: dotnet run 10000 "zipf" "false"

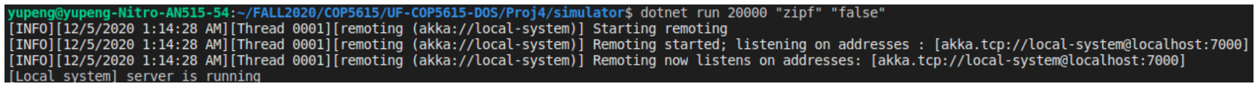
    - NUMBER_OF_USER: An integer indicating the number of simulated users.
    - MODE: A string that should be set to **“zipf”** in this phase.
    - DEBUG: A string that should be set to “false” in this phase

- The terminal will print out the following messages for the server and the simulator respectively to show that the communication path is successfully built:
    - Server:



```
yupeng@yupeng-Nitro-AN515-54:~/FALL2020/COP5615/UF-COP5615-DOS/Proj4/server$ dotnet run
[INFO][12/5/2020 1:08:09 AM][Thread 0001][remoting (akka://remote-system)] Starting remoting
[INFO][12/5/2020 1:08:10 AM][Thread 0001][remoting (akka://remote-system)] Remoting started; listening on addresses : [akka.tcp://remote-system@localhost:9001]
[INFO][12/5/2020 1:08:10 AM][Thread 0001][remoting (akka://remote-system)] Remoting now listens on addresses: [akka.tcp://remote-system@localhost:9001]
[Server] server is running
```

    - Simulator:



```
yupeng@yupeng-Nitro-AN515-54:~/FALL2020/COP5615/UF-COP5615-DOS/Proj4/simulator$ dotnet run 20000 "zipf" "false"
[INFO][12/5/2020 1:14:28 AM][Thread 0001][remoting (akka://local-system)] Starting remoting
[INFO][12/5/2020 1:14:28 AM][Thread 0001][remoting (akka://local-system)] Remoting started; listening on addresses : [akka.tcp://local-system@localhost:7000]
[INFO][12/5/2020 1:14:28 AM][Thread 0001][remoting (akka://local-system)] Remoting now listens on addresses: [akka.tcp://local-system@localhost:7000]
[Local system] server is running
```

C. <u>Note</u>

- the commands should be **run in the folder by the above order** (i.e., first in the server folder and then in the simulator folder), otherwise the simulator will not be able to find the server.
- When the simulator disconnects, which means that all the clients have terminated their processes, there will be some error messages showing up on the server side that I’m still not able to address. But this should not affect the functionality of the engine.
- **The process has to be start from the beginning to run the simulator again**. What I mean by this is, although the server will keep listening to events, the data stored in the previous run will not be cleaned, so **the server process should also be terminated manually (i.e., Ctrl + C or Cmd + C) to start another run**.
- **The folder log has to exist in the first place to save the log file**. The program does not automatically generate the folder. The reason is that I couldn’t figure out how to create a folder with “write” permission. I can only create a new folder with “ReadOnly” attribute, so the workaround is to create the folder named “log” in advance.

2. <u>Dependencies</u>

To accomplish the task, the following additional packages are utilized:

A. <u>Akka.Remote</u>

- Version: 1.4.10
- To allow the communication between two independent processes.

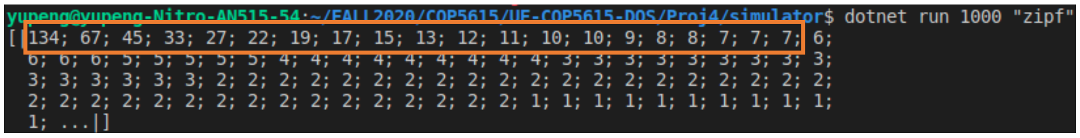
B. <u>FSharp.json</u>

- Version: 0.4.0
- Messages are sent back and forth between two processes in the **JSON format**. This package provides the functionalities of **serializing and deserializing** the JSON format messages.

3. <u>Functionality</u>

The simulator is able to test 11 functionalities. Based on my design, the total number of requests depends on the number of clients, which in my opinion is similar to the real-world case. The following items describe how the numbers of requests are decided for each function, where **N** is the **number of clients**:

- Register users
  - The number of registered users equals **N**.
- Send tweets
  - The number of tweets sent equals **N**.
- Send tweets with hashtags
  - The number of tweets sent with hashtags equals $\left\lceil \frac{N}{4} \right\rceil$.
  - This number is larger than the number of tweets with mentions because I assume that people use hashtags more often than mentioning others based on my personal experience with the real Twitter.
- Send tweets with mentions
  - The number of tweets sent with mentions equals $\left\lceil \frac{N}{10} \right\rceil$.
- Subscribe to celebrities
  - A celebrity is defined as the client who has many subscribers.
  - The number of celebrities is $\left\lceil \frac{N}{50} \right\rceil$.
  - For example, in the Zipf distribution simulation with N = 1000, 20 clients will be assigned as the celebrity and will be followed by a certain number of clients. The specific number is selected from the first 20 entries of the Zipf distribution array calculated based on the formula provided in the previous section:



```
yupeng@yupeng-Nitro-AN515-54:~/FALL2020/COP5615/UF-COP5615-DOS/Proj4/simulator$ dotnet run 1000 "zipf"
[|134; 67; 45; 33; 27; 22; 19; 17; 15; 13; 12; 11; 10; 10; 9; 8; 8; 7; 7; 7; 6;
  6; 6; 6; 5; 5; 5; 5; 5; 4; 4; 4; 4; 4; 4; 4; 4; 4; 3; 3; 3; 3; 3; 3; 3; 3; 3;
  3; 3; 3; 3; 3; 3; 2; 2; 2; 2; 2; 2; 2; 2; 2; 2; 2; 2; 2; 2; 2; 2; 2; 2; 2; 2;
  2; 2; 2; 2; 2; 2; 2; 2; 2; 2; 2; 2; 2; 2; 2; 2; 1; 1; 1; 1; 1; 1; 1; 1; 1; 1;
  1; ...|]
```

- Send Retweets
  - The number of retweets sent equals $\left\lceil \frac{N}{20} \right\rceil$.
  - Retweets are chosen from the tweets received by the client who queries them. The intuition is that before retweeting other people's tweets, the client has to query the tweets first to know exactly which tweet he or she wants to retweet, which is similar to the behavior of surfing other people's tweets first and then retweeting the interesting ones.
  - **Celebrities send more tweets and retweets**.
- Query tweets with hashtags
  - $\left\lceil \frac{N}{4} \right\rceil$ clients are randomly selected to query tweets with a certain hashtag, which is also randomly selected.
- Query tweets with mentions
  - $\left\lceil \frac{N}{10} \right\rceil$ clients are randomly selected to query tweets that mention himself of herself (i.e., my mentions).
- Query the tweets sent by the subscribed users
  - $\left\lceil \frac{N}{10} \right\rceil$ clients are randomly selected to query tweets sent by one of the users that he or she subscribed to.

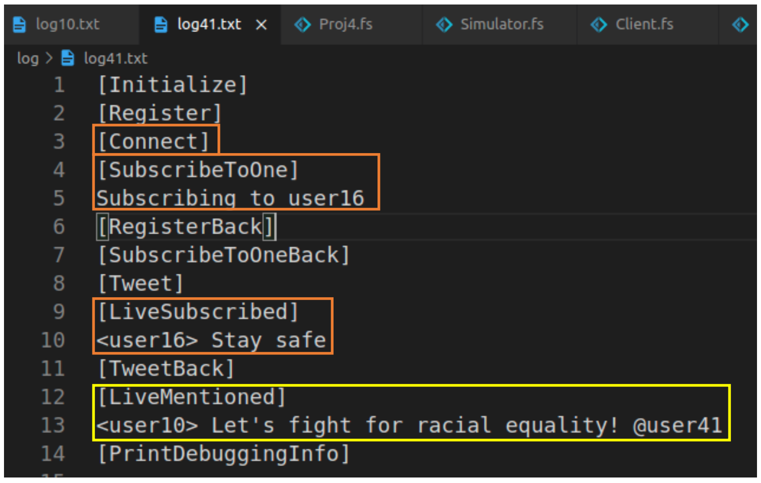
- Connect and Disconnect
  - Upon registering, all the users are by default connected to the server, which is usually the case in the real world.
  - In the simulation, $\left\lceil \frac{N}{10} \right\rceil$ of the clients will disconnect first, and then after simulating some of the other functionalities, $\left\lceil \frac{N}{10} \right\rceil$ of the clients will connect to the server again.
  - When the client is connected, the tweets with the following two conditions will be delivered automatically without querying:
    1) the tweets sent by a subscribed client and 2) the tweets that mention the client.

4. Performance

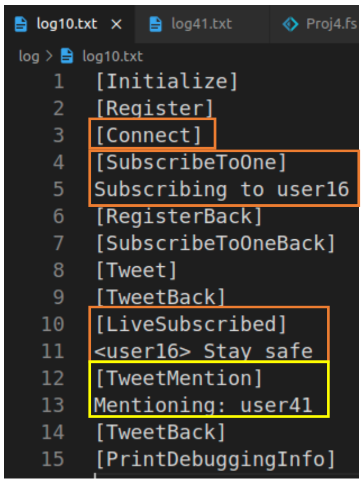
A. Correctness

A small number of users (e.g., 50) is used to test the correctness of the engine API with the help of the debugging files written to the **"log"** folder as described above. We can look at several examples in one of the trials:

- Some general explanations of the symbols
  - Square brackets include the operation that is performed.
  - The <> brackets include the user who tweeted the tweet that follows the brackets.
  - Each client[ID] has the name of user[ID]. In the real-world case, the name user[ID] should be replaced with real names such as Dennis or Alex.
  - As mentioned above, the file name "log[ID].txt" saves the operation log of client[ID].
- Live delivery for subscribed and mentioned
  - As shown in the figures below, in the scenario of 50 people, only one celebrity will be selected, which is **user16** in this case. We can see that client41 and client10 never disconnect from the server, and that they both subscribe to user16. Therefore, they both received the **[LiveSubscribed]** message when user16 tweeted "Stay safe". Also, since client41 is connected when client10 sends tweet with @user41, client41 received the **[LiveMentioned]** message with the tweet: "<user10> Let's fight for racial equality! @user41 "

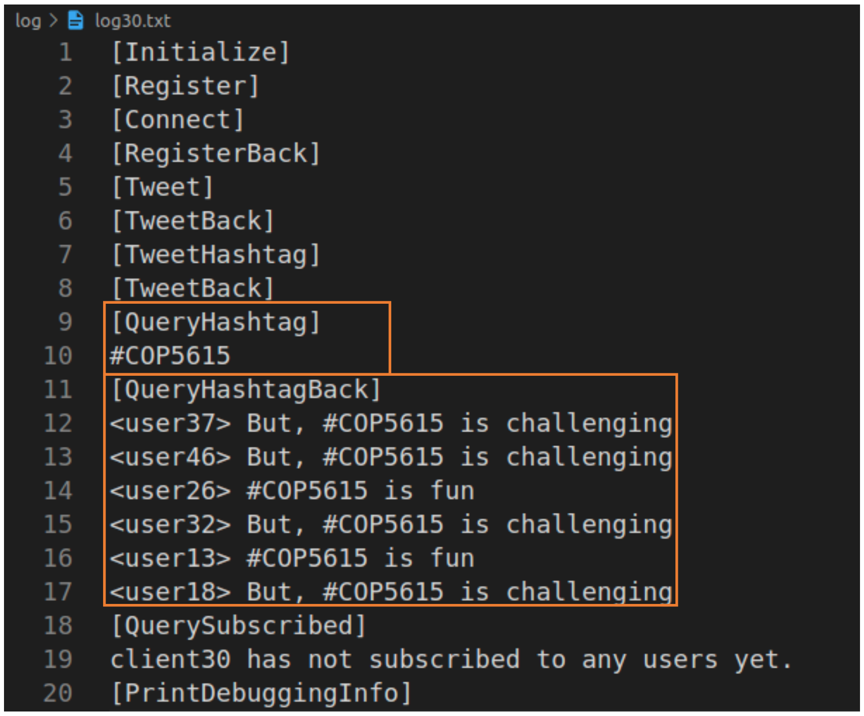


```
[Initialize]
[Register]
[Connect]
[SubscribeToOne]
Subscribing to user16
[RegisterBack]
[SubscribeToOneBack]
[Tweet]
[LiveSubscribed]
<user16> Stay safe
[TweetBack]
[LiveMentioned]
<user10> Let's fight for racial equality! @user41
[PrintDebuggingInfo]
```



```
[Initialize]
[Register]
[Connect]
[SubscribeToOne]
Subscribing to user16
[RegisterBack]
[SubscribeToOneBack]
[Tweet]
[TweetBack]
[LiveSubscribed]
<user16> Stay safe
[TweetMention]
Mentioning: user41
[TweetBack]
[PrintDebuggingInfo]
```

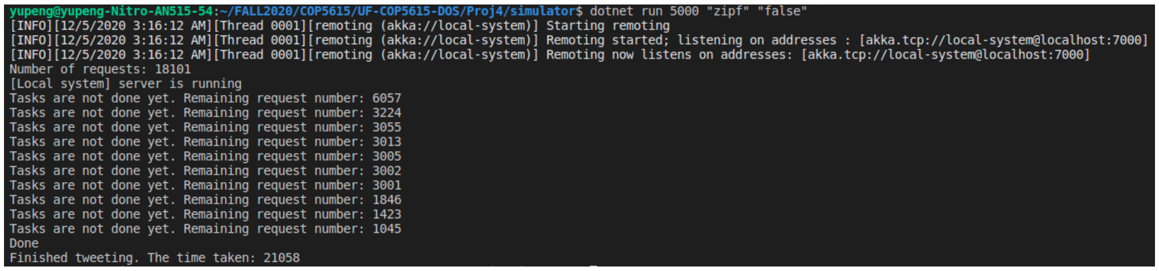
- As shown in the figure below, hashtags (and mentions) can be placed in any positions, which is similar to the real Twitter. The figure also shows the functionality of querying tweets with a specific hashtag, which **is #COP5615** in this case.



```
[Initialize]
[Register]
[Connect]
[RegisterBack]
[Tweet]
[TweetBack]
[TweetHashtag]
[TweetBack]
[QueryHashtag]
#COP5615
[QueryHashtagBack]
<user37> But, #COP5615 is challenging
<user46> But, #COP5615 is challenging
<user26> #COP5615 is fun
<user32> But, #COP5615 is challenging
<user13> #COP5615 is fun
<user18> But, #COP5615 is challenging
[QuerySubscribed]
client30 has not subscribed to any users yet.
[PrintDebuggingInfo]
```

B. Setting

- The order of each simulated functions
  - Register N clients (connected) → subscribe to $\left\lceil \frac{N}{50} \right\rceil$ celebrities → $\left\lceil \frac{N}{10} \right\rceil$ disconnect → N send tweets → $\left\lceil \frac{N}{4} \right\rceil$ send tweets with hashtags → $\left\lceil \frac{N}{10} \right\rceil$ connect → $\left\lceil \frac{N}{10} \right\rceil$ send tweets with mentions → $\left\lceil \frac{N}{4} \right\rceil$ query tweets with a certain hashtag → $\left\lceil \frac{N}{10} \right\rceil$ query my mentions → $\left\lceil \frac{N}{20} \right\rceil$ retweet → $\left\lceil \frac{N}{10} \right\rceil$ query the tweets subscribed to → Done
- **Stopping criteria**
  The simulator keeps track of the total number of requests sent. The server replies the "Back" messages to the client who sent the request after the request is performed. For example, for the **[QueryHashtag]** message, the corresponding "Back" message is the **[QueryHashtagBack]** message. Then, upon receiving the "Back" messages, the clients will notify the simulator to reduce the recorded number of requests. The program stops after all the requests are completed, as shown in the figure below.

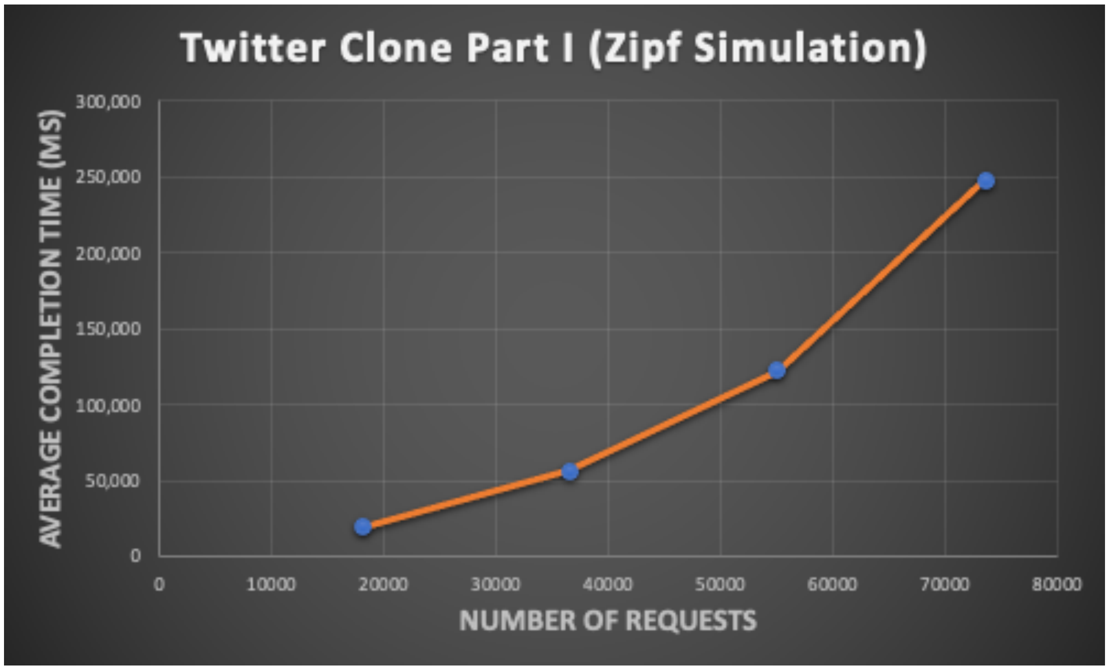


```
yupeng@yupeng-Nitro-AN515-54:~/FALL2020/COP5615/UF-COP5615-DOS/Proj4/simulator$ dotnet run 5000 "zipf" "false"
[INFO][12/5/2020 3:16:12 AM][Thread 0001][remoting (akka://local-system)] Starting remoting
[INFO][12/5/2020 3:16:12 AM][Thread 0001][remoting (akka://local-system)] Remoting started; listening on addresses : [akka.tcp://local-system@localhost:7000]
[INFO][12/5/2020 3:16:12 AM][Thread 0001][remoting (akka://local-system)] Remoting now listens on addresses: [akka.tcp://local-system@localhost:7000]
Number of requests: 18101
[Local system] server is running
Tasks are not done yet. Remaining request number: 6057
Tasks are not done yet. Remaining request number: 3224
Tasks are not done yet. Remaining request number: 3055
Tasks are not done yet. Remaining request number: 3013
Tasks are not done yet. Remaining request number: 3005
Tasks are not done yet. Remaining request number: 3002
Tasks are not done yet. Remaining request number: 3001
Tasks are not done yet. Remaining request number: 1846
Tasks are not done yet. Remaining request number: 1423
Tasks are not done yet. Remaining request number: 1045
Done
Finished tweeting. The time taken: 21058
```

C. Table for the Zipf distribution simulation

dotnet run NUMBER_OF_CLIENTS “zipf” “false”

Number of clients	5,000	10,000	15,000	20,000
Number of requests	18,101	36,504	54,997	73,534
Average completion time (ms)	19,667	56,368	122,923	248,320



D. Maximum number of clients and requests

Based on my design, the maximum number of clients and the corresponding requests are **20,000 clients and 73,534 requests**. When I tried to increase the number of simulated clients, I received the error message of “**oversized payload**”. According to the error message, the messages I tried to send exceeded the maximum allowed size of 128,000 bytes. My hypothesis is that the queries asking for multiple tweets (e.g., querying tweets with a certain hashtag) are the main factors which caused the problem.