COP5615 DISTRIBUTED OPERATING SYSTEM

FALL 2014

PROJECT 4

TEAM MEMBERS:

Neha Singh
Ruchi Desai

STATISTICS GATHERED FROM THE INTERNET ABOUT TWITTER USERS

Total number of twitter users	1,000,000,000
Number of users having 0 to 50 followers	800,000,000
Number of users having 51 to 100 followers	60,000,000
Number of users having 101 to 500 followers	90,000,000
Number of users having 501 to 1000 followers	19,500,000
Number of users having 1001 to 5000 followers	19,500,000
Number of users having 5001 to 1 million followers	10,972,400
Number of people having 1 to 10 million followers	76
Number of people having 10 to 20 million followers	46
Number of people having 20 to 30 million followers	8
Number of people having more than 30 million followers	8
Number of users following 0 to 50 people	740,000,000
Number of users following 51 to 100 people	80,000,000
Number of users following 101 to 500 people	140,000,000
Number of users following 501 to 1000 people	25,000,000
Number of users following 1001 to 5000 people	15,000,000

- 75 percent of all the tweets are tweeted by 5 percent of the users.
- 40 percent of Twitter users do not tweet

IMPLEMENTATION DETAILS (PART 1)

- The simulator requires 6 machines to run, one of the machines being the server and the rest being the clients.

- Server details:

  - On the server machine, there is one master and 5 worker servers. Each worker server will listen to and respond to traffic from one client.

  - On the server machine, we maintain the user-follower mapping as a global Array of Lists. Each element in the array corresponds to the List of followers of a particular user. Each List is generated using idiomatic Scala expression and information from the gathered statistics.

  - The server machine also has Arrays of Queues for the homepages and mentions feeds of the users.
  - Each tweet consists of id of the tweeter, type, name if there is a mention of any user and msg (which is the body of the tweet).

  - Each tweet coming from the users on the client is assumed to be processed and can be of 3 types:
    a. Normal tweet – In which there is no mention of any user.
    b. Mention1 – In which user is mentioned at the start of the tweet.
    c. Mention2 – In which user is mentioned anywhere in the middle of the tweet.

    Depending on the type, the tweet is queued into the homepage and the mentions feed Queues of the respective followers (as per the rules of Twitter)

  - The server is run first and allowed to create the user-follower mapping. It takes about 1 minute to create the user-follower mapping. After that, the clients join the server one by one and user creation is triggered on them by the server.

  - After every 3 seconds, the server machine displays the statistics about tweets and homepage and mentions feed requests received. Our program currently stores the most recent 100 tweets for each user in the homepage Queue and the most recent 50 mentions for each user in the mentions feed Queue.

- Client details:

  - Each client machine creates 20,000 users. A total of 100,000 users were simulated.

  - Based on the statistics gathered, each user has a different tweeting rate based on its id. The same applies to the rate of requesting for homepages and mention feeds. (This has been implemented via the scheduled messages sent by the dispatcher from prestart method).

- Results of running the program on our laptop:

Load Type	Light
Number of tweets	1,801,406
Number of Homepage + mentions feed requests	1,202,033
Total	3,003,439
Duration of run	10 minutes
Rate (requests/second)	5005

Load Type	Heavy
Number of tweets	1,867,542
Number of Homepage + mentions feed requests	5,98,839
Total	2,466,381
Duration of run	10 minutes
Rate (requests/second)	4110

- Results of running the program on a single pcluster server:

Load Type	Light
Number of tweets	1,918,833
Number of Homepage + mentions feed requests	1,282,418
Total	3,201,251
Duration of run	10 minutes
Rate (requests/second)	5335

Load Type	Heavy
Number of tweets	3,205,910
Number of Homepage + mentions feed requests	999,872
Total	4,205,782
Duration of run	8 minutes*
Rate (requests/second)	8762

* Program crashed after 8 minutes.

PROBLEMS ENCOUNTERED

- Creation of user-following mapping takes a long time. Since the total number of users is 100,000, it takes around 30 minutes to insert users into the follower lists of their respective followings if done in a sequential manner. If we try speeding up the process by replacing the sequential lookup by the method used in binary search, only a part of the entire user space is searched. This can be solved using recursion but due to a large number of users, this leads to StackOverflow error. So, currently we have used the idiomatic way of generating lists for the follower mapping.

- The rate of tweets sustained by user had to be initially reduced in order for the program to not produce the “GC overhead limit exceeded error”.

- Another error produced when tweet rate is high is the OutOfMemory error due to Java Heap Space.

CHANGES MADE SINCE FIRST SUBMISSION

- Made number of users paramterized.

- Made request rate parameterized. User will be asked to provide a value for load which can be either light or heavy.

STEPS TO EXECUTE (WITHOUT SBT)

1. Required files:
   - Server.scala
   - Client.scala
   - Common.scala
   - Common.jar

2. Generate Common.jar via sbt using the other 3 files

3. Place the required files mentioned above under the same folder on Linux machine.

4. cd scala/bin (ie the directory where you have stored your scala folder)

5. On machine 1 type `./scala -classpath "<full paths to the jar files that need to be included, separated by :>" <full path to Server.scala> <no of users> <ip_addresses of all client machines separated by spaces>`

6. Wait till machine 1 displays the message "`Server is ready`". On 5 different machines, type `./scala -classpath "<full paths to the jar files that need to be included, separated by :>" <full path to Client.scala> <no of users per client> <ip address of server> <client id> <load>`

   `load` can have values `heavy` or `light`

7. The program will automatically terminate after 10 minutes.

8. The following jar files are required:

```
akka-actor_2.11-2.3.6.jar
akka-remote_2.11-2.3.6.jar
netty-3.8.0.Final.jar
protobuf-java-2.5.0.jar
config-1.2.1.jar
scala-library-2.11.2.jar
common.jar
```

Scala version = 2.11.2
Akka version = 2.3.6
SBT version = 0.13.6

STEPS TO EXECUTE (WITH SBT)

Currently this version has only been tested on a single machine by opening multiple instances. The following steps are written assuming the attached folder named `Twitter_Load_Demo_SBT_Single_Machine` is being used:

1. Open 6 terminal screens

2. On the first screen, cd to Server folder under `Twitter_Load_Demo_SBT_Single_Machine` and type `sbt clean.`

3. On the second terminal cd to Client1, on the third terminal cd to Client2 and so on. Each time type `sbt clean.`

4. In each of the terminals, type `sbt compile`.

5. On the terminal which is in Server folder, type the following command:
   `sbt "run 100000 127.0.0.1 127.0.0.1 127.0.0.1 127.0.0.1 127.0.0.1"`
   where 100000 is the number of users and the parameters after that stand for IP addresses of clients.

6. Wait till the Server terminal shows the message "`Server is ready`". Then on the terminal which is in Client1 folder, type the following command:
   `sbt "run 100000 127.0.0.1 1 light"`
   where, `100000` is the total number of users across all clients
   `127.0.0.1` is the IP address of server
   `1` is the client id
   `light` is the type of load. (load can have values `heavy or light`)
   Do this for the rest of the clients as well, each time putting the appropriate client id in place of `1` and keeping the rest of the parameters the same.

7. The program will automatically terminate after 10 minutes

IMPLEMENTATION DETAILS (PART 2)

- Server has been implemented adhering to REST architecture using Spray Framework with Akka Actors.

- It consists of the following files:

    (i) Main.scala = serves as the main method and creates REST Servers

    (ii) SprayServer.scala = uses RoundRobinRouter to create multiple instances of ServerActor and sends the appropriate message to ServerActor depending on the URL entered.

    (iii) ServerActor.scala = handles and completes requests (incoming tweets and homepage and mentions feed requests)

    (iv) Data.scala = contains functions for creating user-following mappings and initializing homepage and mentions feed for all users.

    (v) Messages.scala = contains the messages passed to ServerActor

    (vi) Entries.scala = contains conversion protocol to json format

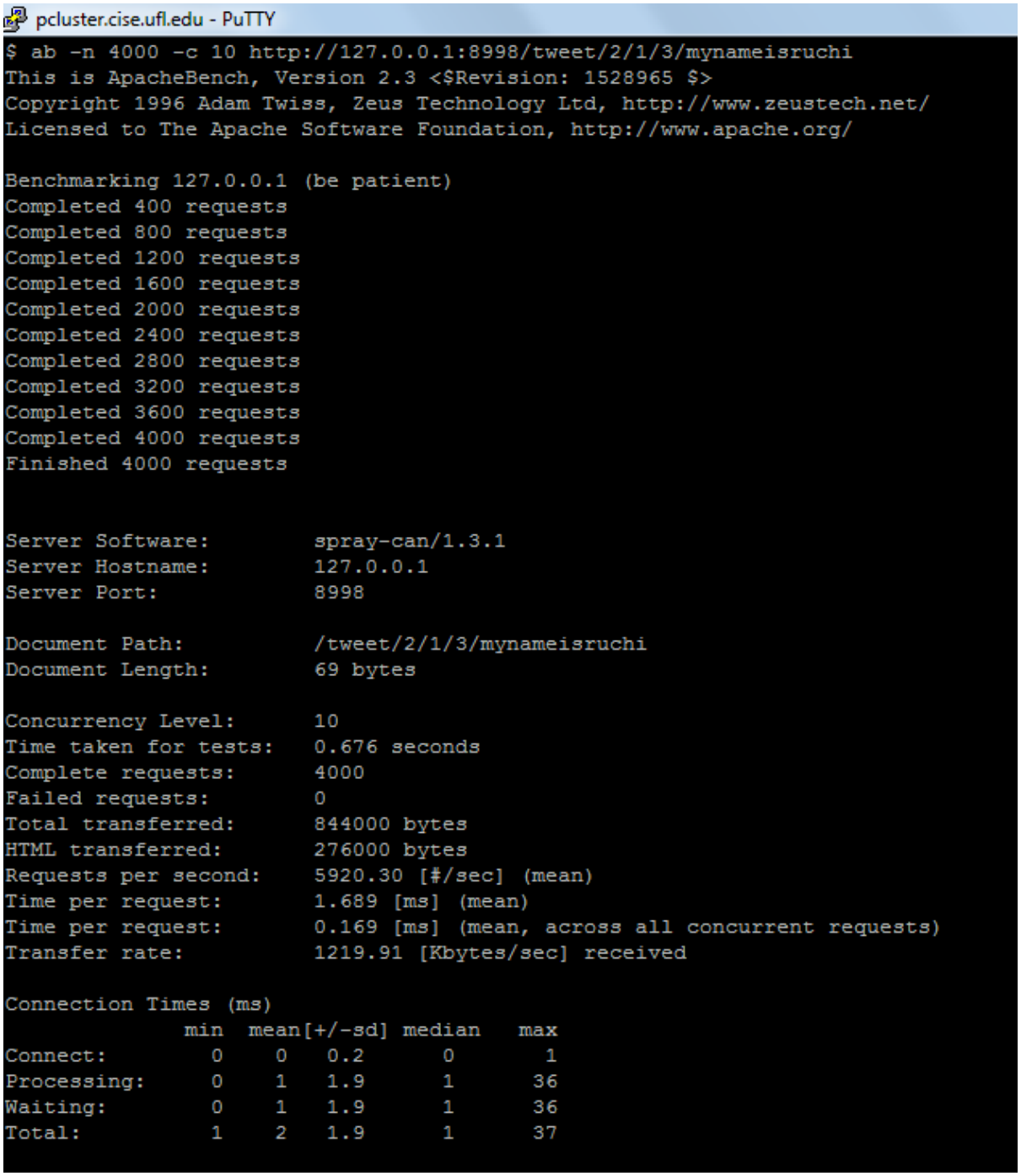
- The following parts of Twitter API have been implemented:
    - Tweet
      e.g.: http://127.0.0.1:8998/tweet/1/0/0/hello

    - View Homepage
      e.g.: http://127.0.0.1:8998/homepage/2

    - View Mentions Feed
      e.g: http://127.0.0.1:8998/mentionsfeed/2

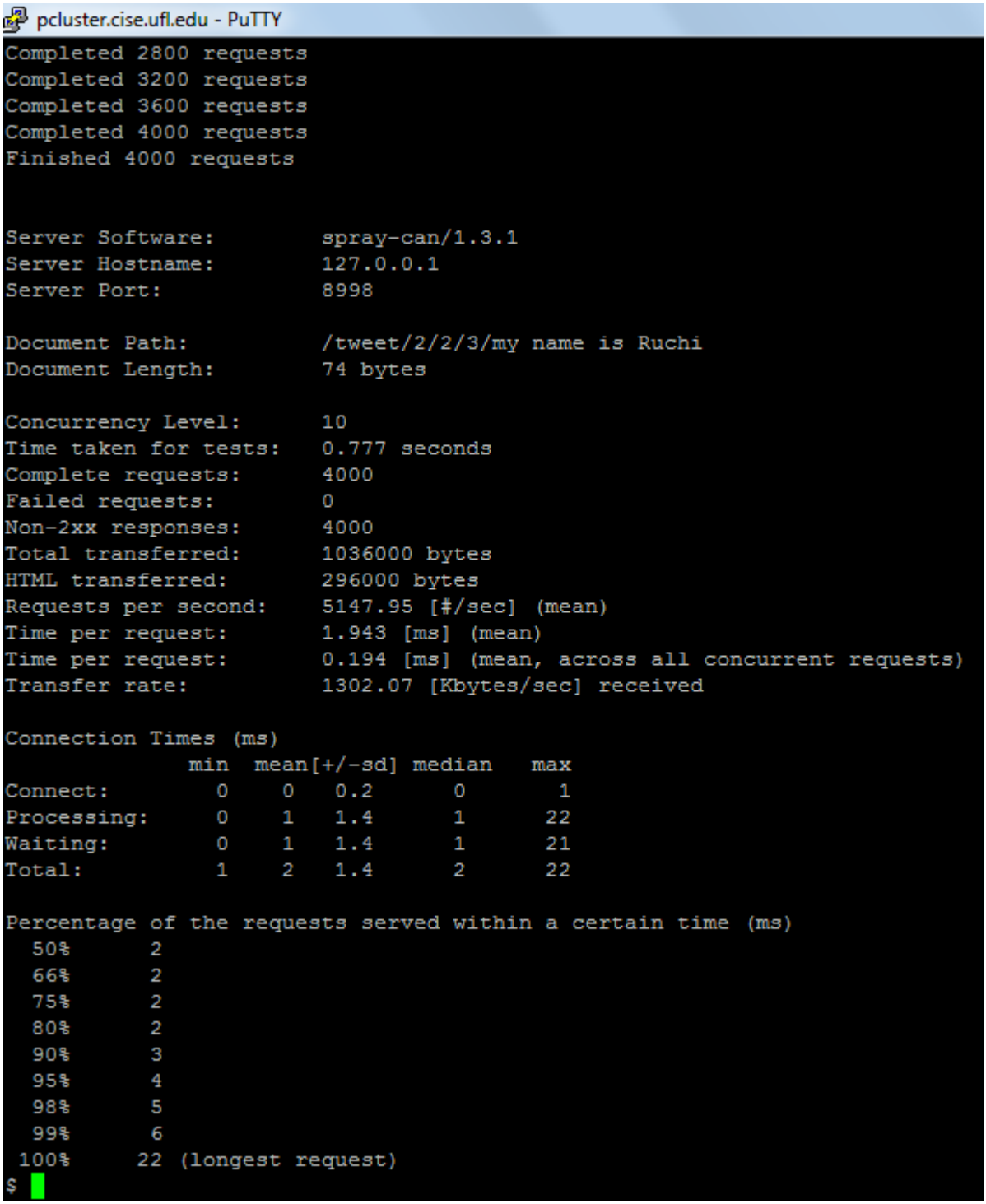
    - View Followers
      e.g.: http://127.0.0.1:8998/followers/3

RESULT OF BENCHMARKING AGAINST APACHE BENCHMARKING TOOL ON PCLUSTER SERVER



pcluster.cise.ufl.edu - PuTTY

```
$ ab -n 4000 -c 10 http://127.0.0.1:8998/tweet/2/1/3/mynameisruchi
This is ApacheBench, Version 2.3 <$Revision: 1528965 $>
Copyright 1996 Adam Twiss, Zeus Technology Ltd, http://www.zeustech.net/
Licensed to The Apache Software Foundation, http://www.apache.org/

Benchmarking 127.0.0.1 (be patient)
Completed 400 requests
Completed 800 requests
Completed 1200 requests
Completed 1600 requests
Completed 2000 requests
Completed 2400 requests
Completed 2800 requests
Completed 3200 requests
Completed 3600 requests
Completed 4000 requests
Finished 4000 requests


Server Software:        spray-can/1.3.1
Server Hostname:        127.0.0.1
Server Port:            8998

Document Path:          /tweet/2/1/3/mynameisruchi
Document Length:        69 bytes

Concurrency Level:      10
Time taken for tests:   0.676 seconds
Complete requests:      4000
Failed requests:        0
Total transferred:      844000 bytes
HTML transferred:       276000 bytes
Requests per second:    5920.30 [#/sec] (mean)
Time per request:       1.689 [ms] (mean)
Time per request:       0.169 [ms] (mean, across all concurrent requests)
Transfer rate:          1219.91 [Kbytes/sec] received

Connection Times (ms)
              min  mean[+/-sd] median   max
Connect:        0    0   0.2      0       1
Processing:     0    1   1.9      1      36
Waiting:        0    1   1.9      1      36
Total:          1    2   1.9      1      37
```


pcluster.cise.ufl.edu - PuTTY
Completed 2800 requests
Completed 3200 requests
Completed 3600 requests
Completed 4000 requests
Finished 4000 requests
Server Software: spray-can/1.3.1
Server Hostname: 127.0.0.1
Server Port: 8998
Document Path: /tweet/2/2/3/my name is Ruchi
Document Length: 74 bytes
Concurrency Level: 10
Time taken for tests: 0.777 seconds
Complete requests: 4000
Failed requests: 0
Non-2xx responses: 4000
Total transferred: 1036000 bytes
HTML transferred: 296000 bytes
Requests per second: 5147.95 [#/sec] (mean)
Time per request: 1.943 [ms] (mean)
Time per request: 0.194 [ms] (mean, across all concurrent requests)
Transfer rate: 1302.07 [Kbytes/sec] received
Connection Times (ms)
min mean[+/-sd] median max
Connect: 0 0 0.2 0 1
Processing: 0 1 1.4 1 22
Waiting: 0 1 1.4 1 21
Total: 1 2 1.4 2 22
Percentage of the requests served within a certain time (ms)
50% 2
66% 2
75% 2
80% 2
90% 3
95% 4
98% 5
99% 6
100% 22 (longest request)
$

PROBLEMS ENCOUNTERED

- Initially we used a single ServerActor for handling the requests. This led to very low rates for requests processed per second (7 requests/second). So, in order to increase the number of requests handles per second, we introduced RoundRobinRouter. When the SprayServer creates an instance of ServerActor, it attaches to the ServerActor a RoundRobinRouter so that the specified number of instances of ServerActor are created and messages are distributed among them in round-robin manner. We found the number 50000 (of routes) optimal for the command:
  ab -n 4000 -c 10 http://127.0.0.1:8998/tweet/2/1/3/hello

- After the introduction of RoundRobinRouter, we started getting the error:
  …tail of empty list… on commands similar to:
  ab -n 4000 -c 10 http://127.0.0.1:8998/tweet/1/0/0/hi (ie type 0 tweets)
  This error could have resulted due to use of mutable data structures for storing homepage and mentions feed entries. We were using mutable Queue which do not provide the proper support for shared access. The issue was resolved and did not appear after we started using the immutable Vector for storing homepage and mentions feed entries. As mentioned in Scala documentation, Vectors provide support for shared access and will not allow access to more than one entity at the same time.

- We were unable to test our Part 1 code on multiple pcluster servers. We could only test it on a single pcluster server by opening multiple instances. However, the same code runs on multiple lin116 servers (without using sbt).

STEPS TO EXECUTE (WITH SBT)

The following steps are written assuming the attached folder named `SprayTwitterOnLaptop` is being used:

1. In Terminal, cd to the above mentioned folder.
2. Type `sbt clean`.
3. Type `sbt compile`.
4. Type `sbt run`.
5. Wait till the message "`Server is ready`" is displayed.
6. Open a separate Terminal
7. Type `curl -v http://127.0.0.1:8998/tweet/1/0/0/hello` to tweet
8. Type `curl -v http://127.0.0.1:8998/homepage/1` to view homepage

Meaning of different commands:

Type 0 tweet: http://127.0.0.1:8998/tweet/1/0/0/hello = user with id 1 has tweeted the message "hello".

Type 1 tweet: http://127.0.0.1:8998/tweet/2/1/3/3iscool = user with id 2 has tweeted the message 3iscool where 3 is mentioned at the beginning.

Type 2 tweet: http://127.0.0.1:8998/tweet/4/2/5/hello5 = user with id 4 has tweeted the message hello5 where 5 is mentioned later in the tweet.

Homepage request: http://127.0.0.1:8998/homepage/2 = will return the homepage entries for user with id 2.

Mentions Feed request: http://127.0.0.1:8998/mentionsfeed/3 = will return the mentions feed entries for user with id 3.

View Followers: http://127.0.0.1:8998/followers/2 = will return the ids of followers of 2.

APPENDIX

Following are the attached files:

`Twitter_Load_Demo_SBT_Single_Machine` = Code for running Part 1 of Project 4 on a single machine (by opening multiple instances)

`Twitter_Load_Demo_Multiple_Machines` = Code for running Part 1 of Project 4 on multiple machines (without using sbt)

`SprayTwitterOnLaptop` = Code for running Part 2 of Project 4

`SprayTwitterOnBigServer` = Code for benchmarking Part 2 of Project 4 on a pcluster server