COP 5615 Distributed Operating System Principles

Project 4 – Part 2

Date - 11/27/2014

# TEAM MEMBERS –

Divya Ramachandran (46761308)

Mugdha Kumar (54219168)

Palak Shah (55510961)

# INTRODUCTION –

As Part 2 of the course project for Distributed Operating Systems Principles class, we have simulated the server backend of Twitter as an http web service using spray IO. We simulated various characteristics of the Twitter Client base by studying the various statistics about Twitter posted officially by Twitter and from other sources online.

The functionality of Twitter was simulated as follows, keeping the original design as the reference. The server machine hosts many actors that distribute the work of storing and fetching tweets. These actors make use of various tables to keep track data related to users. We divide the load by assigning equal number of users to each server worker actor.

For message passing, the JSON format has been used as it enables lightweight messages.

# TWITTER-SERVER –

The server side is made up of many actors arranged in a two-tier hierarchical system. The httpservice actor at top level listens to requests and route them to the actors at next level (called ServerAssigner) where these requests are handled. The total number of users are equally divided among ServerAssigners. ServerAssigners maintain many kinds of tables **for the users assigned to it** – tables of -

1. A follower list for those users
2. Queues that hold last 100 tweets to be displayed on each of those User’s timelines
3. A queue containing tweets where the user has been mentioned
4. A list of messages received by each user

It also holds a table of tweets list mapped by the hashtags they contain.

Functionalities implemented:

1. Tweet (text that is less than 141 characters)
2. Request a user’s timeline
3. Retrieve tweets where a user has been mentioned
4. Retrieve tweets related to a hashtag
5. Retrieve direct messages intended for a user
6. Retrieve user’s followers’ list

All the calculation is done when a tweet arrives at the Server so that different timelines/lists for the user are precompiled and kept ready. Followers of the user who tweeted are identified. Their timeline tables are populated. When a user asks for particular information, the ServerAssigners simply return the corresponding entry from the table.

# TWITTER-CLIENT –

The client side of this projects tries to simulate all users of Twitter. The total number of users is divided by the number of Client Machines (henceforth referred to as “Clients”). The users on the same Client machine have contiguous UserIDs. Using this fact, we can simulate various equations based on UserIDs to simulate actual twitter statistics more closely

The client code also has a hierarchical structure with one actor at the top and all user actors at second level. The top actor is created only for the purpose of initialization and graceful termination and does not cause a bottleneck at Client side (i.e. it does not send any messages).

Each user runs a scheduler to determine when to tweet and when to ask for timeline. At a point after the start of the code, we have simulated the occurance of an event that causes incoming tweets to sky-rocket.

# SIMULATION –

In this project, we have tried to simulate twitter statistics very closely. We have assigned serially incrementing userIDs to our users. This helped us to use various equations in order to simulate real world statistics.

1. We simulated the distribution of tweets per user as per the graph depicted in Figure 1 and described in [1]. It is clear from the graph that a very small number of users generate a large number of tweets. Majority of the users generate very few tweets per day. To simulate this, we used an exponential decay function for userID. Our graph of userID vs Tweets is shown in Figure 2. As you can see, in our code also, very few users are generating most of the tweets while most of the users generate very few tweets

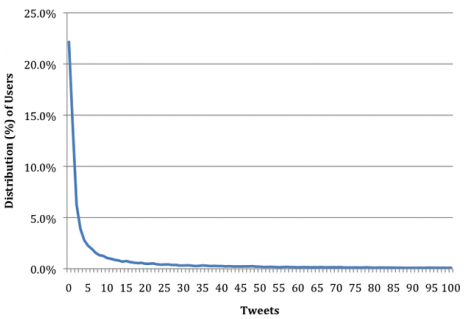


Figure 1 Number of Users vs Tweets found on-line[1]

Figure 2 Users vs Tweets in our Project

1. The number of tweets coming in at Twitter Server remains approximately constant. But sometimes, during a major event, the number of tweets per seconds increases way above the average throughput. One such scenario is described in [2]. Figure 3 depicts how there is a peak in tweets throughput for a short period of time. We have used a scheduler to schedule an event such that at that point, every user in our system will tweet. This is depicted by the steep spike in our throughput graph.

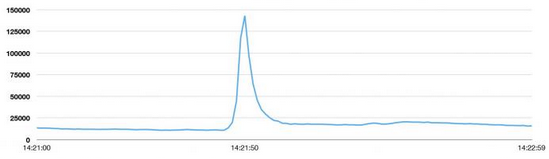


Figure 3 Spike in Twitter Throughput spike during an event[2]

1. As we can accommodate different clients (different client machines), client actors (users) from a single machine can represent users from a particular geographical region. Keeping such an analogy in mind, we have initiated the tweeting at slightly varying times to indicate the effect of different time zones in different geographical locations where daytime can be considered the time of higher activity.
2. The number of followers for different users vary.

# RESULTS –

Results for a total of 10,000 users on two client machines are as follows. The number of tweets and requests remain fairly constant.

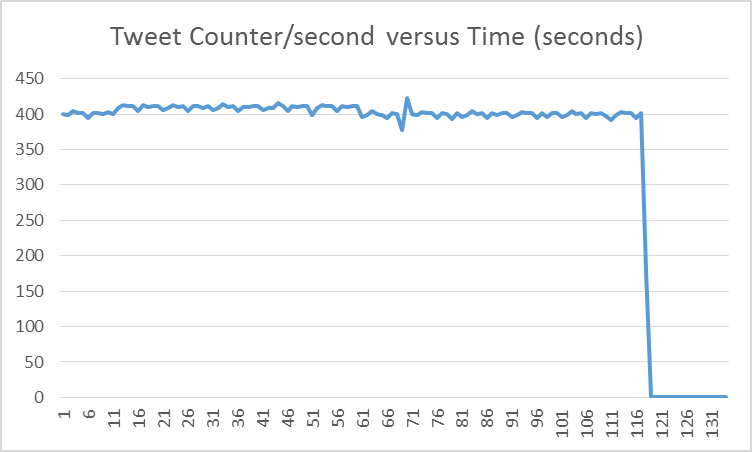


Figure 4 Average tweets/second observed across time

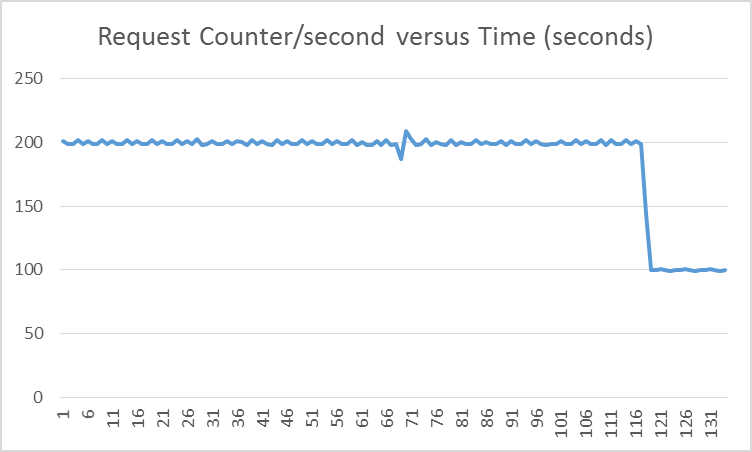


Figure 4 Average tweets/second observed across time

The throughput can be calculated in terms of tweets per second by –

Average tweet counter/second + (average request counter/second \*length of the list of tweets returned)

For the above case of 10000 users on 2 machines, we observed a throughput of about 20000 tweets/second.

The dip in the end is generated because of termination of server and clients.

# REFERENCES –

1. http://www.boxuk.com/blog/twitter-user-demographics/
2. <https://blog.twitter.com/2013/new-tweets-per-second-record-and-how>
3. http://www.beevolve.com/twitter-statistics/#b1