Project Report: Facebook Rest API

Server Endpoints

The Facebook Rest API Server consisted of the following endpoints:

1) Post: user
   Accepts object of type User in json format and creates a new user profile. Sends a conflict status code if user already exists.

2) Delete: user/userid
   Deletes the user based on given userid.

3) Get: users
   Marshalls the list of users and sends the list to the client side where the list of user objects is unmarshalled.

4) Post: addfriend/senderid/receiverid
   This request is made when the user with id ‘senderid’ adds user with id ‘receiverid’ as a friend. The receiverid gets added to the friend requests sent list of the sender and the senderid gets added to the pending friend requests list of the receiver.

5) Post: acceptrequest/senderid/receiverid
   This post request is made when the sender with id ‘senderid’ accepts the request of receiver with id ‘receiverid’. The senderid is removed from the friend request sent list of the receiver and the receiverid is removed from the pending friend request list of the sender. The appropriate ids are added in both the users friend list and they become friends.

6) Get: page/pageid/senderid/receiverid
   The user with id ‘senderid’ requests page of user with id ‘receiverid’. If sender is a friend of receiver and the pages of receiver contains the page with id ‘pageid’ a list of posts contained on that page is sent back to the client sender.

7) Get: friendlist/senderid
   This request can be made from any user to any other user and they do not have to be friends. It returns the list of all the friends of a particular user.

8) Post: acceptallrequests/userid
   This post requests accepts all the pending requests of a particular user. We added this to make simulation easier as simulating accepting one request at a time would be an arduous task.

9) Post: posting/senderid/receiverid/pageid
   This endpoint is for posting a Post on a certain page. The sender posts a post on receiver's page with id 'pageid'. The post is received by unmarshalling the json to Post object.

10) Get: getprofile/userid
    This endpoint returns the use profile which is essentially all the user information like name, age, etc. It sends the data by marshalling the User object into json.

11) Post: upload/image
    This request uploads an image to the server from the client side.

12) Post: creategroup
    Accepts object of type Group in json format and creates a new group. Sends a conflict status code if group already exists. It unmarshalls json data to Group object type.

13) Post: joingroup/userid/groupid
    This request is made when user with id 'userid' wants to join group with id 'groupid'. If user is already a member of the group a conflict message is sent otherwise the user is added to the group.

14) Post: posting/group/userid/groupid
    A user with id 'userid' posts on a group page with id 'groupid'. A user is only allowed to post on the group page if the user is part of the group otherwise a conflict message is sent back to the client.

Client Simulator Model

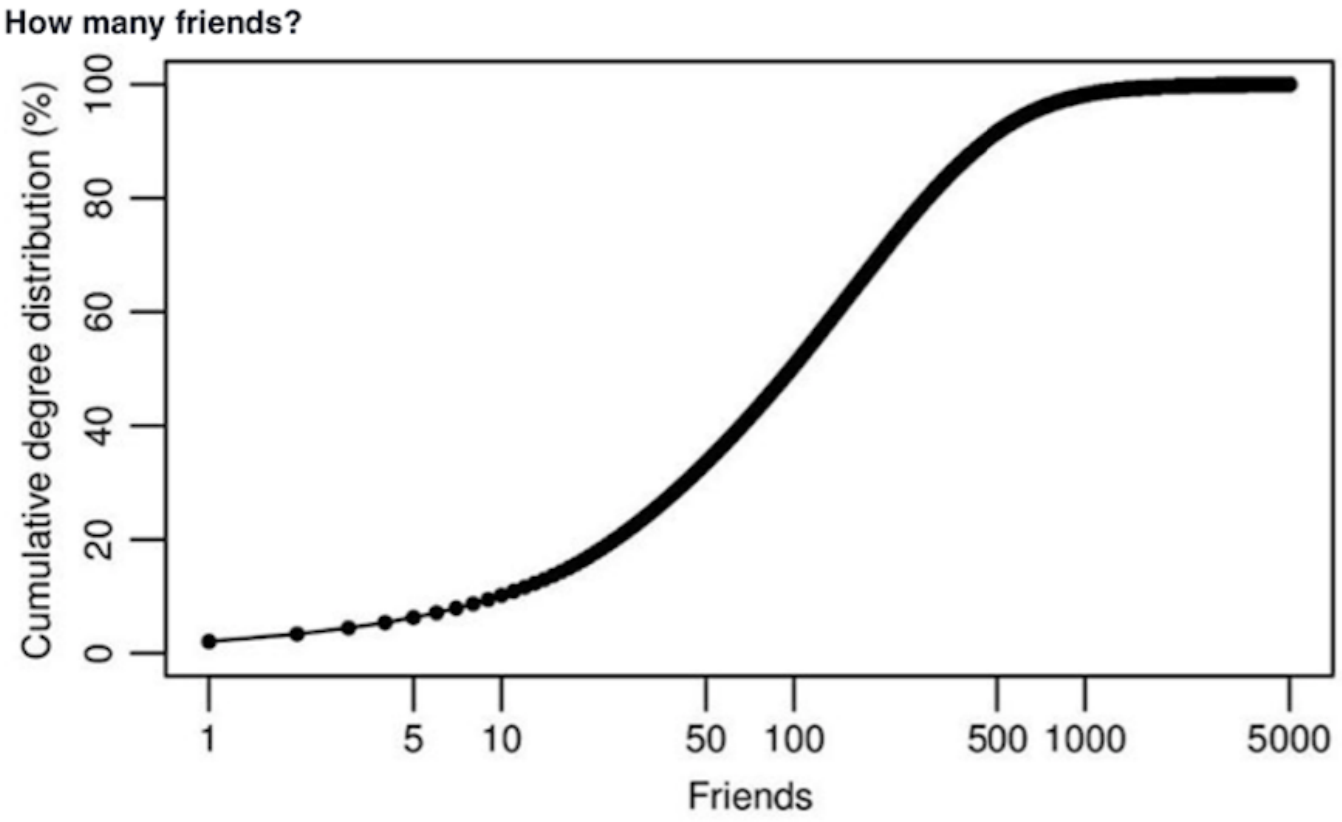


Figure 1 [1]

The above chart shows the cumulative degree of distribution of the number of friends. As we can see about 50% of people have more than 190 friends. We divided the users into 2 groups based on the data from the above chart. The first group has lesser number of friends compared to the second group. Hence, the first group are relatively less active compared to the second group. To maintain the mean number of friends in the above graph (which is about 190 friends) the first group of users have an average of 100 friends while the second group has an average of about 300 friends.

The users in our model were tested by simulating about a 1000 users and each of them interact with all their friends by retrieving and posting information. The friend requests passed between them were independent friend requests, i.e. they come in sequentially. We implemented this to test the message passing speed of the REST API. The above system takes about 40 seconds to create which includes all the initializations and message passing.

Number of rest messages exchanged during this time (assuming reply from the server to be an independent message)

=

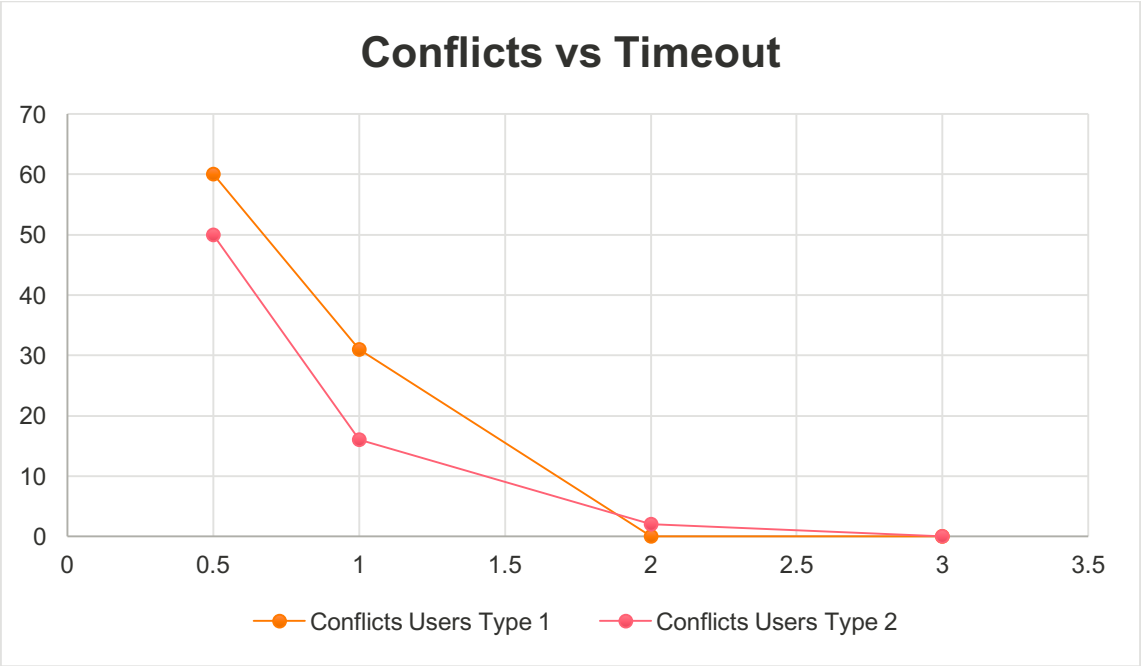
(1000 * 2){creating users} + (200000 * 2){sending friend requests} + (1000 * 2){accepting friend requests}

Total Number of messages exchanged = 404000

Failure Model Testing

Conflict in this situation occurs when a user is unable to post. We increased the timeouts and checked the percentage of users which were in conflict. Higher the timeout, more time used for posting and hence less chances of conflict. As per our findings we chose to use a timeout of 3 as it was a good tradeoff between efficiency and accuracy.

USERS OF FIRST TYPE (5000 Users)			
Percentage	Number of Conflicts	Timeout	smaller
60%	3006	0.5	
31%	1530	1	
0%	0	2	
0%	0	3	
USERS OF FSECOND TYPE (15000 Users)			
Percentage	Number of Conflicts	Timeout	bigger
50%	7498	0.5	
16%	2460	1	
2%	340	2	
0%	0	3	


Conflicts vs Timeout
70
60
50
40
30
20
10
0
0
0.5
1
1.5
2
2.5
3
3.5
Conflicts Users Type 1
Conflicts Users Type 2

References

1) https://www.facebook.com/notes/facebook-data-team/anatomy-of-facebook/10150388519243859

2) https://developers.facebook.com/docs/graph-api

Project Members

Anupam Bahl 36399021 anupambahl@ufl.edu

Siddhant Deshmukh 36568046 siddhantdeshmukh@ufl.edu