Project 4 part 1

Twitter Clone

➢ Team Members

1. Nikhil Chopra (UFID - 98973831)
2. Kanika Gupta (UFID - 96977046)

➢ How to Run

a. Start the server

Go to - project4part1/Server/twitterserver

Run Command - escript twitterserver

This will generate twitterserverIP on the terminal.

b. Start the client

Go to - project4part1/Client/twitterclient

Command - escript twitterclient {twitterserverIP}

➢ Generate Results

To see the results, press Y and enter on the server to generate zipf_results.csv (Excel file) in the folder project4part1/Server/twitterserver. This file will populate with number of followers corresponding to each user having follower count >= 1.

Follow the below steps to generate zipf graph -

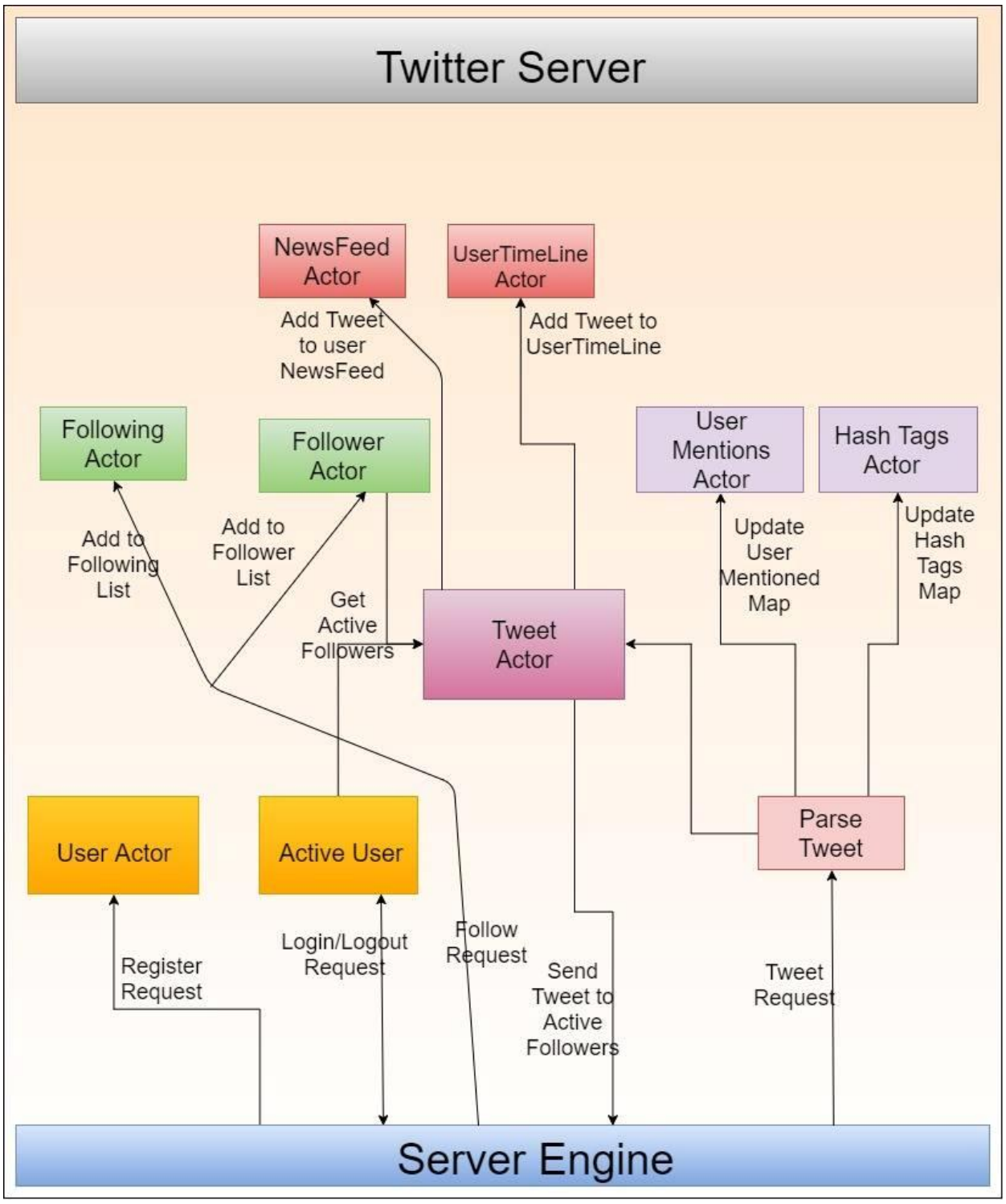
1. Open project4part1/Server/twitterserver/zipf_results.csv in MS Excel.
2. Select column A.
3. Go to Data in the menu bar and sort the results from Z -> A (Decreasing order)
4. Select column A.
5. Go to Insert in the menu bar and select 2-D Line chart to see the zipf distribution of followers.

➢ Twitter Clone Architecture

Project 4 part 1 consists of two separate mix projects – twitterclient and twitterserver.

Twitter Server Architecture

Twitterserver implements server side, wherein in-memory tables are created for each type of list and client requests such as login, logout, tweet, retweet and follow are handled.



This implementation of Twitter Server includes a separate GenServer actor for each feature. These actors have their own ETS table (in-memory table) to store the data.

To access one particular ETS table, the server will call the appropriate GenServer actor to access the table.

The actors along with their ETS tables and usage are listed as under –

Usage	Actor name	ETS Table name	ETS Table structure
User Registration	Users	usersTable	{username, password, fullname}
Login/ Logout	ActiveUsers	activeUsersTable	{username}
Follower Data (A user is followed by whom)	Followers	followerMap	{username, followerListSet}
Following Data (Who is user following)	Followings	followingMap	{username, followerListSet}
NewsFeed Data	Newsfeed	newsFeedMap	{username, tweetsList}
User's tweets list	UserTimeLine	userTimeLineMap	{username, tweetsList}
Tweets	Tweets	tweetsMap	{tweetId, tweet}
Query mentioned tweets	UserMentions	userMentionsMap	{mentionedUser, tweetIdsList}
Query HashTags	HashTags	hashTagsMap	{hashtag, tweetIdsList}

All these actors handle calls which query their respective ETS tables.

There is a **worker actor** in the server, which handles all the requests coming from the client, passes on the requests to the appropriate actors listed above and does all the computation.

- When **Register Request** comes from the client, worker calls Users actor to add that user into the usersTable.
- When **Login Request** comes from the client, worker calls ActiveUsers actor and add that user to the activeUserstable and calls NewsFeed, UserTimeLine actors to send updated newsfeed and usertimeline to the user.
- When **Logout Request** comes from the client, worker calls ActiveUsers actor and remove that user from the activeUserstable.
- When **Tweet Request** comes from the client, worker does the following –
    - Calls Tweets actor to add that tweet in tweetsMap after generating the tweetId.
    - Parses the tweet for any mentioned users and hashtags.
    - If mentioned users are there, worker calls UserMentions actor, for each mentioned user, to add that tweet to the list of that user's mentioned tweets.
    - If hashtags are there, worker calls HashTags actor, for each hashtag, to add this tweet in the list of that hashtag.
    - Calls Followers actor to get the followers of the user who tweeted.

    - Calls ActiveUsers actor to get the list of active users and finds the followers who are active at that time by the intersection of followerList and activeUsersList.
    - For the online followers, the tweet is sent to them live and added to their NewsFeedTable on the server. For this, worker calls NewsFeed actor, for each online follower, to add that tweet in the newsfeedTable of the follower.
    - For the offline followers, the tweet is added to their Newsfeed on the server. For this, worker calls NewsFeed actor, for each offline follower, to add that tweet in the newsfeedTable of the follower. When this follower comes online, he'll be getting the updated list of tweets.
- When **Follow Request** (user1 wants to follow user2) comes, worker calls Followers actor for user2 to add the user1 to the follower list of the user2. Worker also calls Following Actor for user1 to add the user2 in the following list of user1.
- When **Retweet Request** comes, the worker calls Tweets actor to add that tweet in the list of tweets and then calls Followers Actor to find the followers, calls ActiveUsers actor to get active users at that time and send the re-tweet to the live users without querying, Otherwise, the retweet is added to the offline user's newsfeed at the server.

Twitter Client Architecture

The client architecture is divided into following services -

- **User Service -**

The Client Simulator simulates 100 users after 3 seconds. Client sends login request for each of the newly created users. After this, these users are put into 5 categories to achieve zipf distribution.

- Category 1 users – These are the most followed users and have the most tweets.
- Category 2 users
- Category 3 users
- Category 4 users
- Category 5 users - These are the least followed users and have the least tweets (0 most of the times).

- **Follow Service -**

This service will send Follow Requests to the server for each category, the difference being the sleep times for each one. Category 1 users will be followed by other users most frequently than other category's users. This service runs recursively after some sleep time.

- **Tweet Service –**

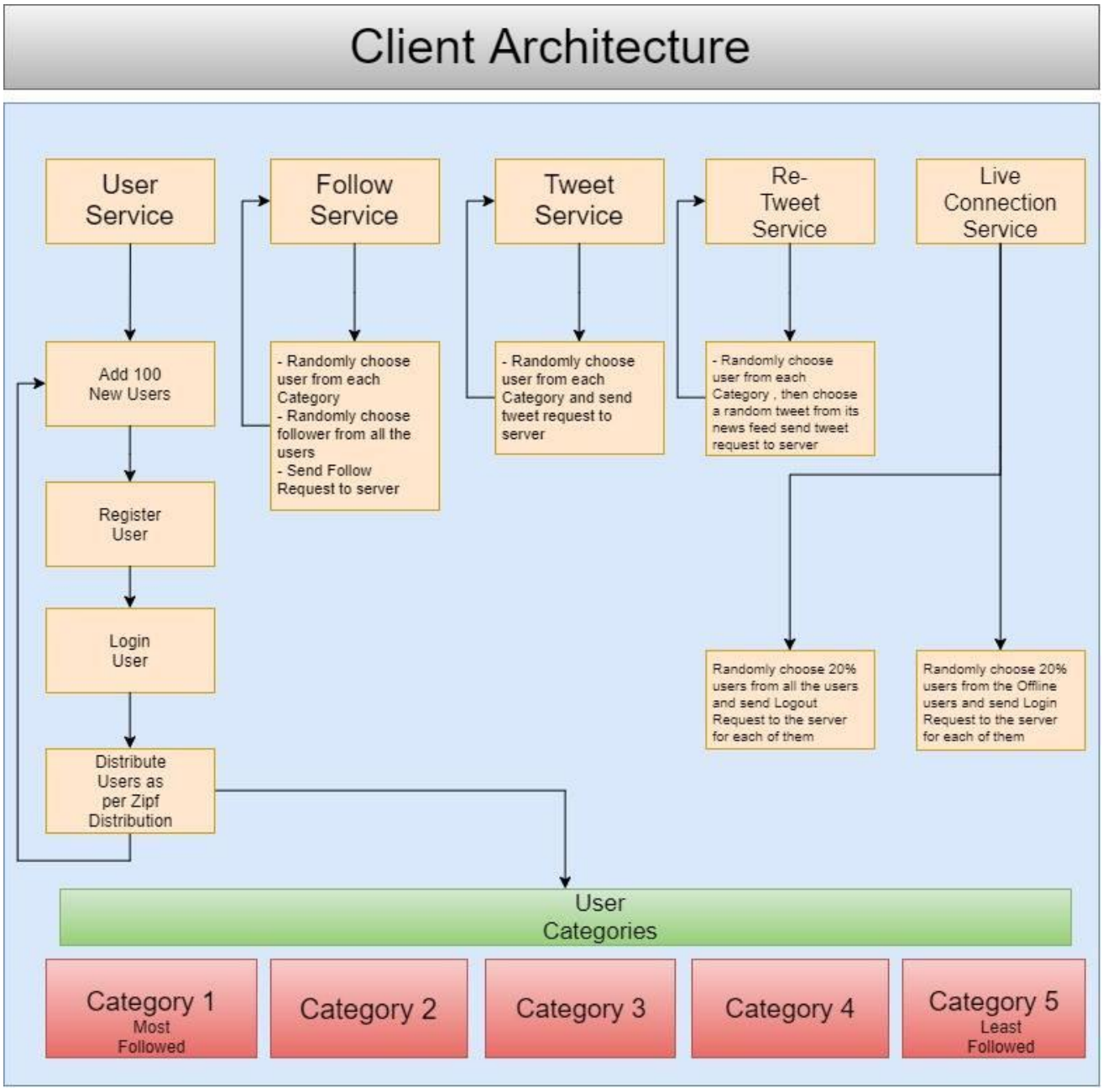
This service will send Tweet Requests to the server for each category, the difference being the sleep times for each one. Category 1 users will tweet most frequently than other category's users. This service runs recursively after some sleep time.

- **Retweet Service –**

This service will send Re-Tweet Requests to the server for each category, the difference being the sleep times for each one. Category 1 users will re-tweet most frequently than other category's users. This service runs recursively after some sleep time.

- **Live Connection Service –**

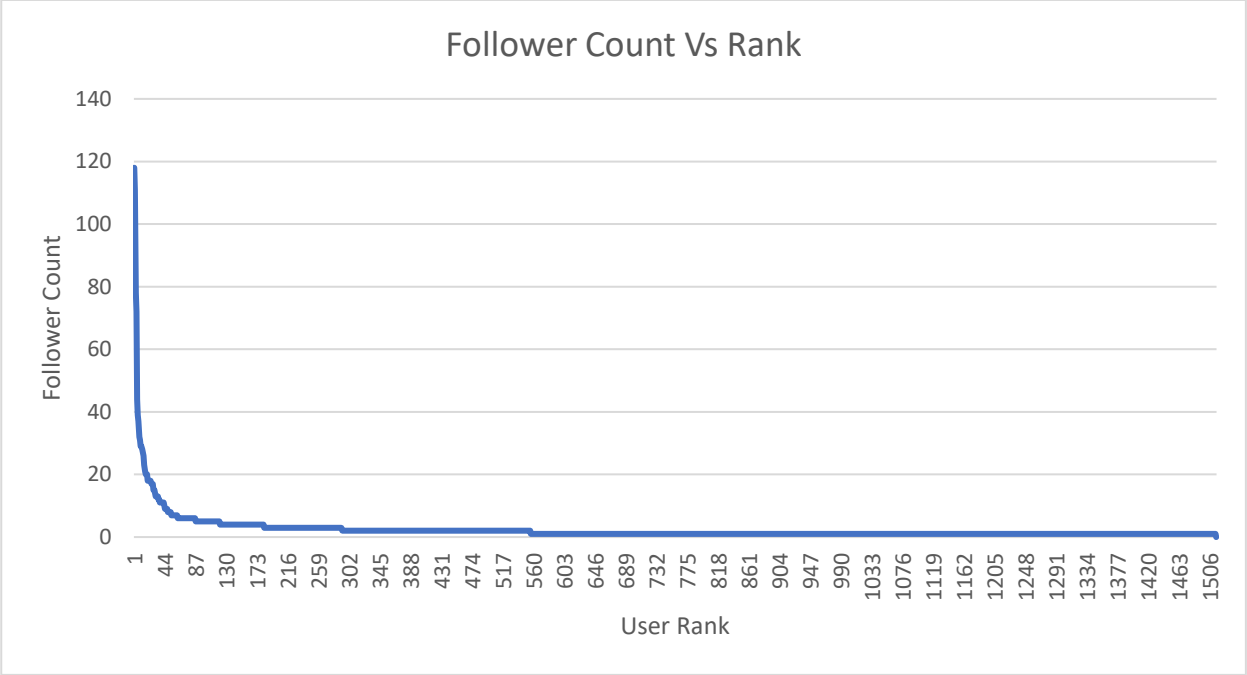
This service handles logging in and logging out of the users. It maintains a set of offline Users. It randomly selects 20% users from the total users and sends Logout request to the server and puts those users in the offline set. After some sleep time, it again selects 20% users from the set of offline users and sends Login request to the server and removes them from offline users set. This service runs recursively after some sleep time.



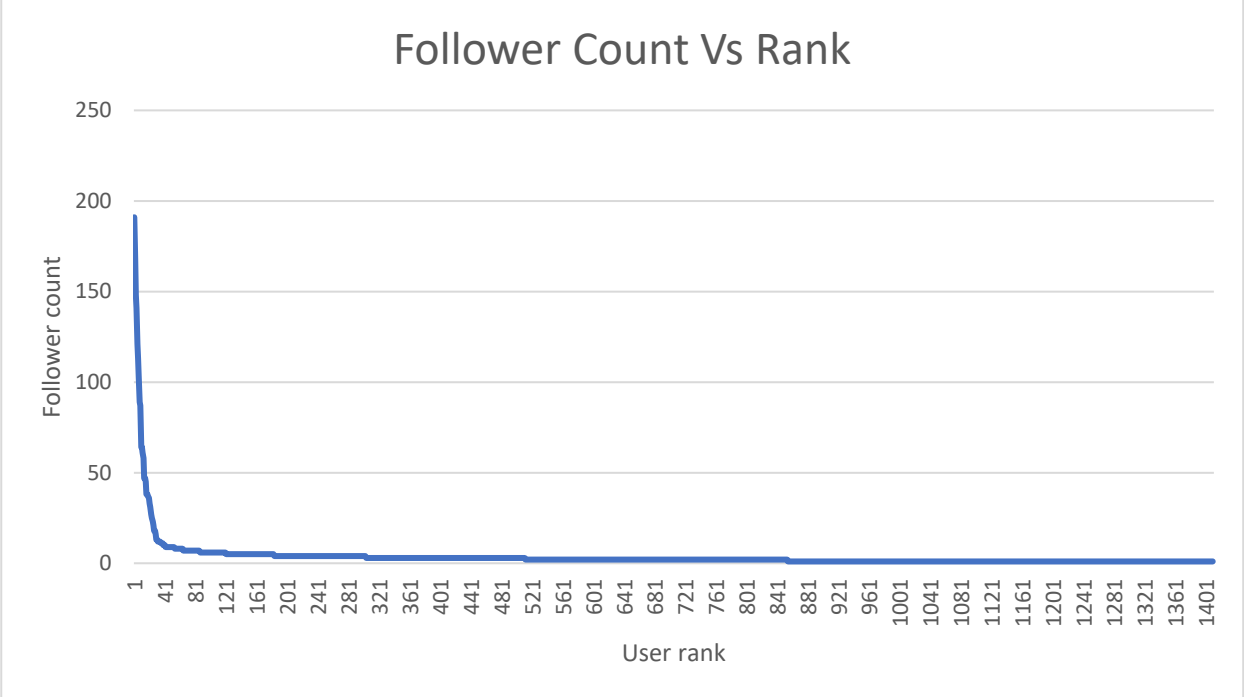
Performance with Zipf distribution

We were able to simulate **10,000 users** running successfully and following zipf distribution of followers. The operations were performed by the simulated users simultaneously are - tweet, follow, login, logout and retweet.

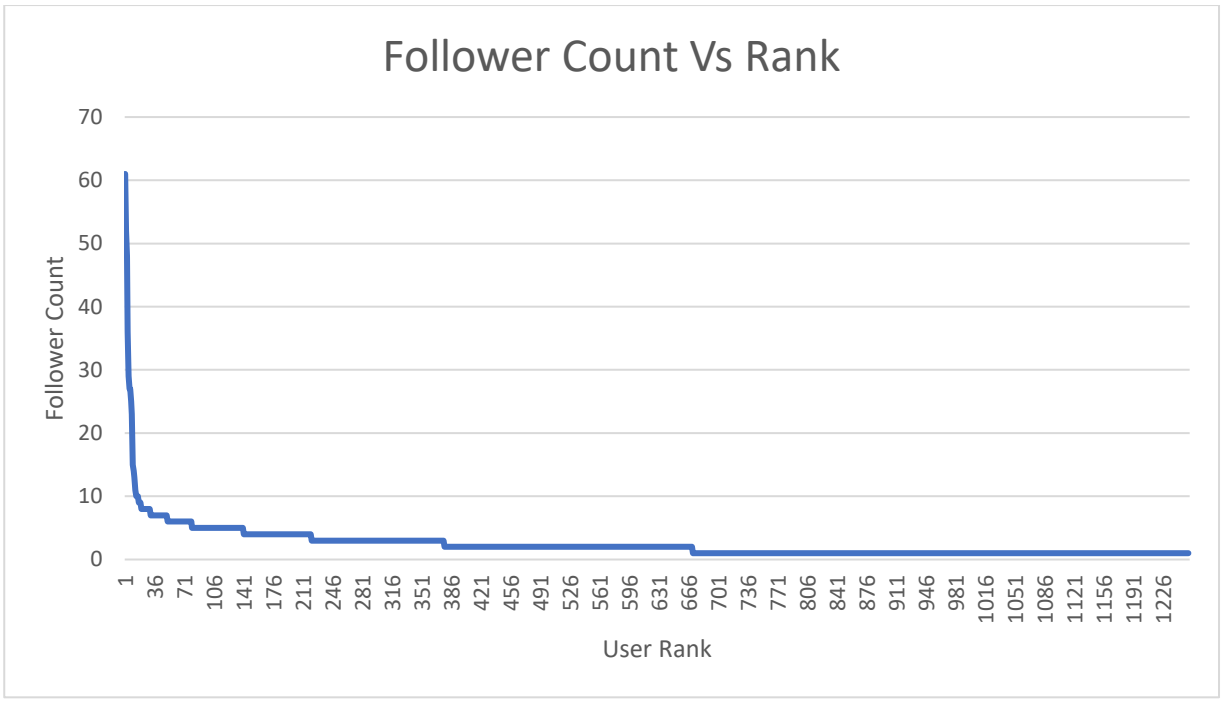
For **7365** users, below is the zipf distribution graph for the follower count of users having followers >1



For **6583** users, below is the zipf distribution graph for the follower count of users having followers >1



For **4000** users, below is the zipf distribution graph for the follower count of users having followers >1



For **3000** users, below is the zipf distribution graph for the follower count of users having followers >1