COP5615 – Distributed Operating System Principles

Project Report - 4 (Part 1)

Team Members

1. Akshay Ganapathy (UFID - 3684-6922)
2. Kamal Sai Raj Kuncha (UFID - 4854-8114)

Input

The input to the program is the server IP address, the port of the server machine where the server engine is currently running, and the number of clients the machine is to handle.

Output

The output is the performance of various aspects of the simulation on the client side such as get tweets and retweets, get mentions, get hashtags.

Zip File Contents

The zip file consists of the readme.md file, Project_Report.pdf file, proj4_server.fsx file and the proj4_client.fsx file, which contains the code to be run.

How To Run

Run and start the server first using the following commands:

Server : dotnet fsi –langversion:preview proj4_server.fsx

After the server program displays 'Server started', then run the following command.

Client : dotnet fsi –langversion:preview proj4_client.fsx <server_ip> <server_port> <number_of_clients>

where 'server_ip' is the IP address of the server,
'server_port' is the Port number in which the server is running, and
'number_of_clients' is the number of clients the machine is to handle.

Languages used

F# was used to code the project.

Platforms used for running the code

Visual Studio Code
.NET version 5.0
NuGET Akka.NET v1.4.25

Basic Requirement: Finished

Implementation

Server

It handles all the requests from the client side actors and disseminates the work to different actors to undertake different operations and finally, sends back the output to the client. The operations performed by the server can be divided into further categories:

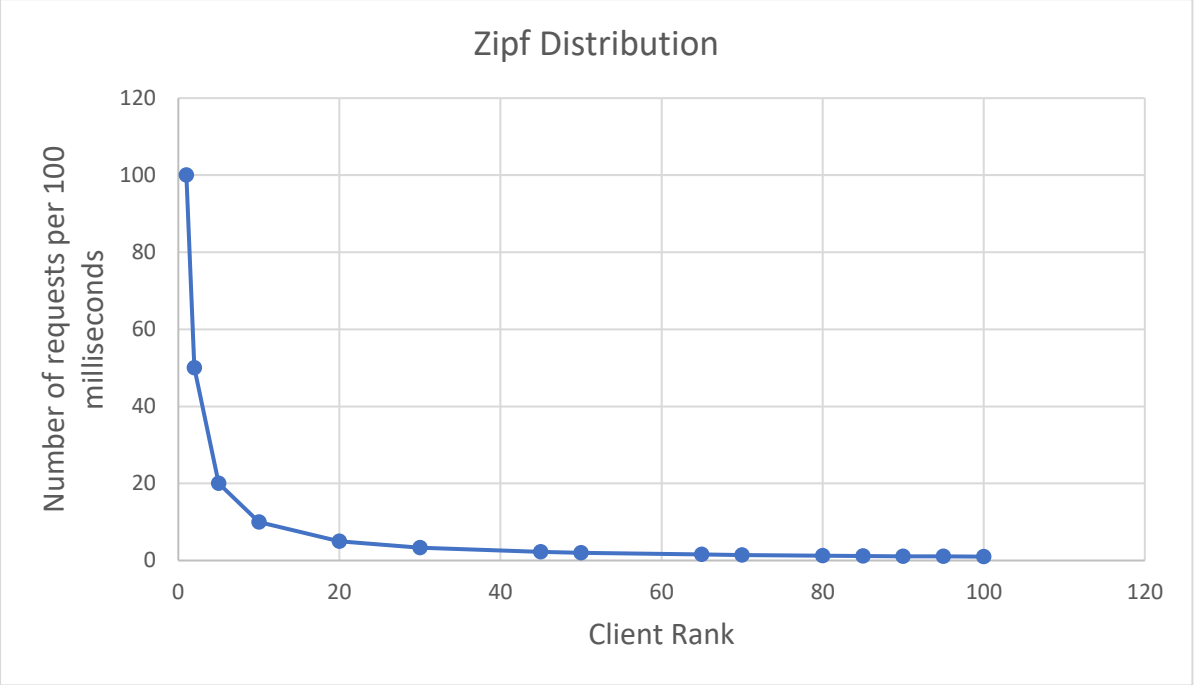
- Requests resolver: Creates actors that accept all the requests from the client side actors and distributes the work to various actors or specific functionalities depending on the type of request received. It also prints the figures and statistics after every 10000 requests.
- Registration resolver: Creates registration actors to perform actions such as login, logout and registration requests from the client. Each of these handlers have distinct modules that help with the functionalities of the handler.
- Followers resolver: Carries out functionalities that include adding followers to the client and sending tweets to all of the followers of the client.
- Tweets handler: Handles the functionalities related to new tweets and retweets. Each of these functions are implemented and carried out by the actors created by TweetsActor function. It receives its input from the client by means of a ServerActor and then creates actors based on the distinct function to be performed.
- TweetsResolver function: Creates actors and carries out functionalities such as parsing tweets, hashtag extraction, and resolving mentions by other users.
- Server handler: This is the main server handler that binds together all of the other functions and actors. It retrieves all the tweets, mentions and hashtags and outputs the statistics for the required data.

Client

It works as a simulator that handles all the requests and sends and receives the requested data from the server. It creates clients and performs functionalities such as registration, adding subscribers, sending the tweets etc. The client referred to here is equivalent to a user who can tweet, login, logout, register, follow, subscribe, and get tweets. By making use of the Zipf distribution, the client is able to perform the above functionalities.

- Zipf distribution: It is a mathematical distribution that is used when there are multiple types of data to be studied, by establishing a relation between rank order and the frequency of occurrence of a statistic. Each client (ranked from 1 to N, the number of clients) makes 1/rank number of requests/ms to the server.
- The number of followers to each client is inversely proportional to the rank of the client.
- For every 100 requests, we would get a login and a logout request.
- For every 10 requests, one is a retweet.
- For every 1000 requests, one would have a get tweet and a mention

Plot - Zipf distribution



Execution table

Number of Clients	Time taken to solve following number of requests (ms)				
	All requests (for every 10000 requests)	Retweets and tweets (for every 100 requests)	Get Tweets (for every 100 requests)	Get Mentions (for every 100 requests)	Get HashTags (for every 100 requests)
10	35.7646	86.5787	66.44	140.745	55.8
50	42.9219	196.4758	357.295	384.8	400.68
100	44.0185	390.6493	744.76	804.48	1285.84
500	62.6003	4125.1003	1106.752	1071.71	3878.86
1000	108.0611	7013.808	1720.67	4058.34	2567.54
2000	139.1246	25053.962	2211.89	4341.9	2665.972

Largest Network Managed

On running the code, the largest number of clients we were able to work with was 2000. To work with even larger clients would have led to us obtaining results but with time consuming delays. In addition, we also managed to run the program by connecting the server with multiple clients (3) and it worked as expected.

Plot - Number of Clients v Runtime for total requests