# Distributed Operating System Principles (COP5615)

## Project-4 Part 1

Zhaoyang Chen (UFID: 9013-7235)

## 1. Project Overview

Build a scalable Twitter-like engine under the idea of actor model with Erlang language. The engine supports the functionality of accounts registration, publishing tweets, subscribing to other users, re-tweet and query tweets (tweets with specific hashtags and tweets in which the user is mentioned).

## 2. Architecture

The twitter-like service will store information about users, and their subscriptions to other users ("followers"). The tweets will contain the text (limited to 140 characters) and together with a timestamp. The whole service data can be described as the following diagram:

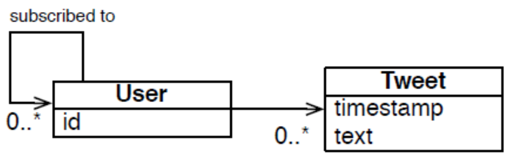

Fig 1: User and Tweets diagram.

The project includes the following four files, corresponding to four different module functionalities. In the project, I use multiple independent client processes that simulate thousands of clients and a single-engine process.

***server.erl:*** The interface defines the interaction of the engine system.

***single_actor.erl***: A single Erlang process to implement the service.

***workload_simulator.erl***: Generate some workload: number of users, number of subscribers for each user, times for each action (tweets, get tweets, etc.), and also some random text to publish.

***benchmark.erl***: Test the CPU running time, find the best and worst scenarios when the engine deploying on large scale of samples.

## 3. Simulation

The entire testing output can be found at the Readme files, in the code directory.

- ***Running Environment***

  Memory: 3.8GB

  Processor: 11th Gen Intel® Core™ i5-1135G7 @ 2.40GHz × 4

  OS: Ubuntu 20.04.5 LTS / 64-bit

  Virtualization: VMware

- ***Simulate as many users as you can***

In this scenario, generate certain number of users and also, we generate certain number of followers for each user. Test the running time for these generations. As for the specific kind of actions such as tweets and get tweets. I repeat the same action for a certain amount of time to check the total running time.

| Cases | Data Scale / Wall Clock Time | | |
|---|---|---|---|
| | Num of Users | Num of followers for each user | Number of each action |
| Test 1 | 10 / (0.001s) | 1 / (0.001s) | 1 / (0.000s) |
| Test 2 | 100 / (0.157s) | 10 / (0.464s) | 10 / (0.321s) |
| Test 3 | 500 / (1.995s) | 50 / (8.788s) | 25 / (2.552s) |
| Test 4 | 1000 / (9.093s) | 100 / (176.561s) | 50 / (28.885s) |

- ***Simulate periods of live connection and disconnection for users***

I test the total running time for program to generate 5000 users and with randomly 10 subscribers. Here is exactly 5 separate running time in benchmark test.

| Test index | 1 | 2 | 3 | 4 | 5 |
|---|---|---|---|---|---|
| CPU Time | 5135ms | 5179ms | 5756ms | 5656ms | 5643ms |
| Wall Clock Time | 1525ms | 1527ms | 1729ms | 1684ms | 1321ms |

- ***Largest number of users I simulated***

I have simulated a total of 2000 users. The program will return desire output.

Note: In the program, I write some code to generate random readable text for users to publish, which take most of the running times. If this part of the time is removed, the actual running time will be significantly reduced and the problem size can reach tens of thousands. Just approaching the deadline :(