COP5615 Project 4: Implementing a Twitter-like engine using Actor Model in Erlang

Documentation

Srinivas Koushik Kondubhatla (UFID: 69238911)
Dharani Kanchanapalli (UFID : 75351996)

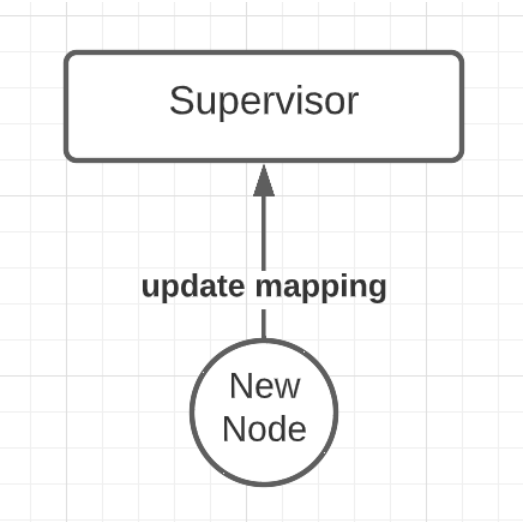
Problem Statment

Implementation of twitter-like engine using actor-model in Erlang. The main functionalities of this engine are

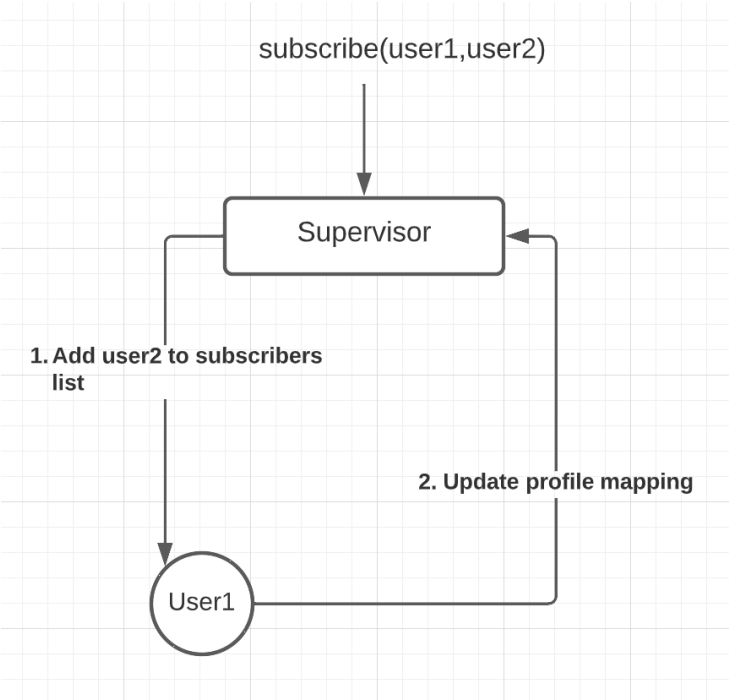
- Register
- Tweet
- Retweet
- Deliver tweets live(if possible)
- Query tweets by subscribed user.
- Query tweets by Hashtag
- Query tweets by Mentions
- Subscribe

Architecture

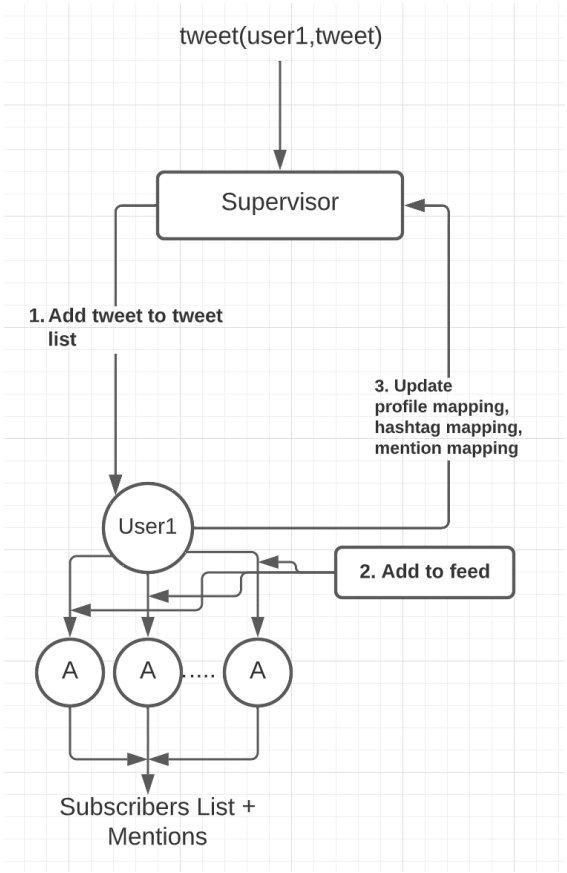
Register



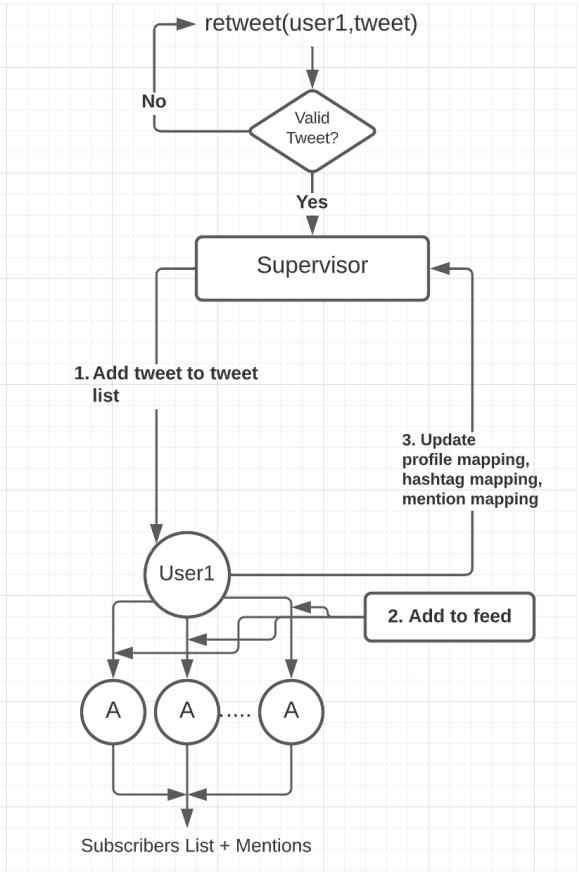
Subscribe



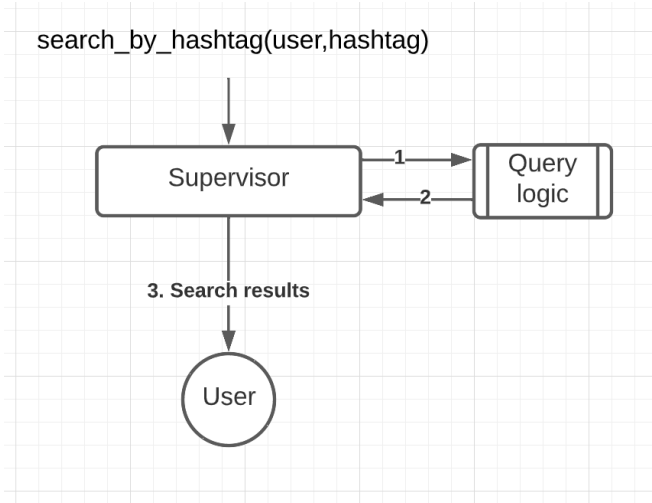
Tweet



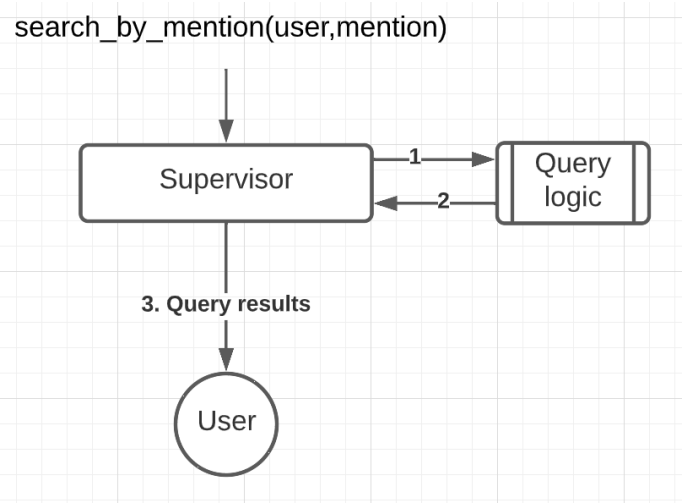
Retweet



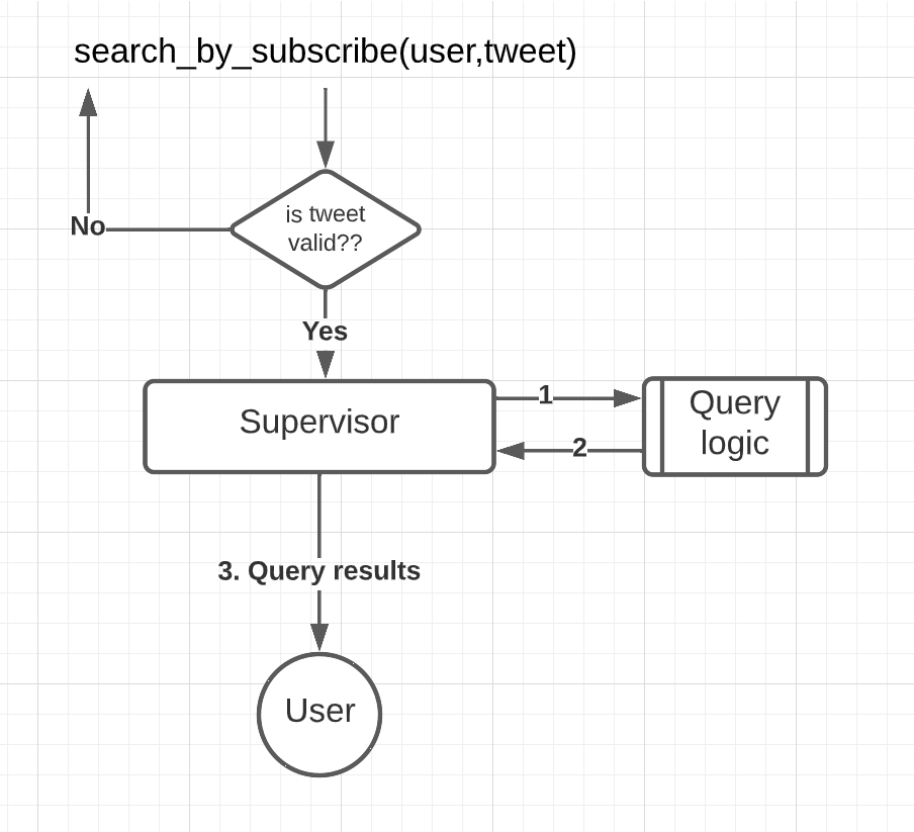
Query Tweet by hashtag



Query Tweet by hashtag



Query Tweet of subscribed user



Responsibilities

Profile Structure

```
Profile = #{"server" => Server_Id},
Profile_Username = maps:put("username", Username, Profile),
Profile_Password = maps:put("password", Password, Profile_Username),
Profile_Email = maps:put("email", Email, Profile_Password),
Profile_Tweet_List = maps:put("tweets", [], Profile_Email),
Profile_Subscription = maps:put("subscriptions", [], Profile_Tweet_List),
Profile_Feed = maps:put("feed", [], Profile_Subscription),
Profile_Id = maps:put("id", Pid, Profile_Feed),
```

Supervisor Responsibilities

- Store updated
  - Username - Profile mapping
  - Hashtag - Tweets mapping
  - Mention - Tweets mapping
- Compute query results and send the message to respective user.
- Redirect all requests to appropriate user.

User Responsibilities

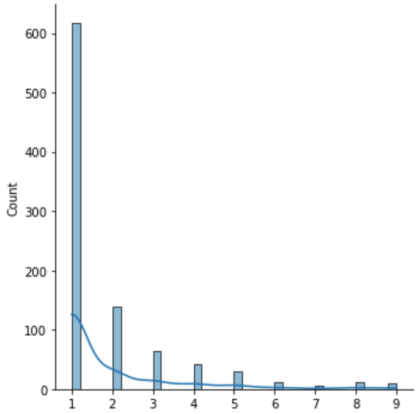
- Store Updated profile
- Update tweet, subscription, feed lists
- Send the tweet to subscribers.
- Live delivery of tweets to connected users
- Compute Hashtags, Mentions in the tweet.
- Display search query results.

Note: The maximum number of users that are active in the network is tested upto 10,000. At the 50,000 mark, the laptop started to hang.

Zipf Distribution (zeta distribution)

Zipf Law :frequency of the words $\propto$ 1/priority rank

Distribution(histogram plot)



Run Instructions

Compile files

- c(helper).
- c(twitter).
- c(client).

Get Supervisor Id

- Id = twitter:get_server().

Generate Users

- L = helper:helper_get_usernames(1000,[],Id).

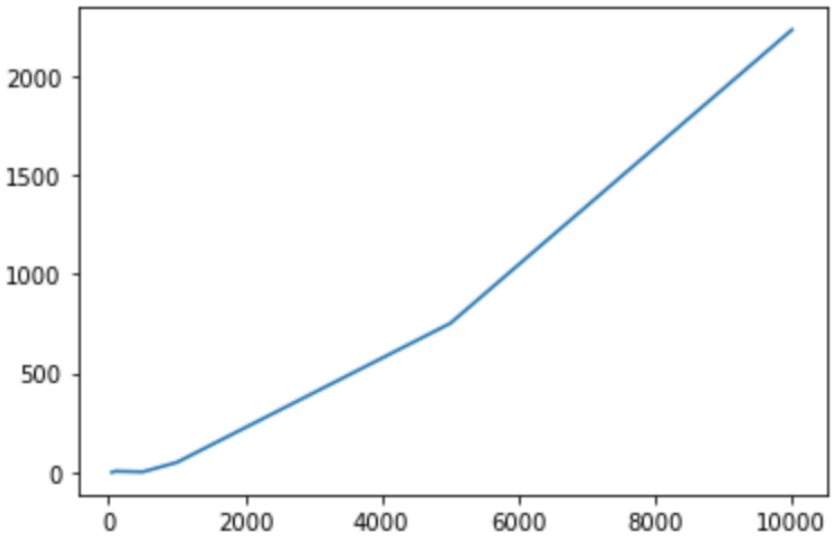
Client functions

```
Used 0 times | Cannot extract specs (check logs for details)
tweet(Username, Tweet, Server_Id) ->…

Used 0 times | Cannot extract specs (check logs for details)
retweet(Username, Tweet, Server_Id) ->…

Used 1 times | Cannot extract specs (check logs for details)
register(Username, Password, Email, Server_Id) ->…

Used 0 times | Cannot extract specs (check logs for details)
subscribe(Username1, Username2, Server_Id) ->…

Used 0 times | Cannot extract specs (check logs for details)
search_by_hashtag(Username, Hashtag, Server_Id) ->…

Used 0 times | Cannot extract specs (check logs for details)
search_by_mention(Username, Hashtag, Server_Id) ->…
```

Results

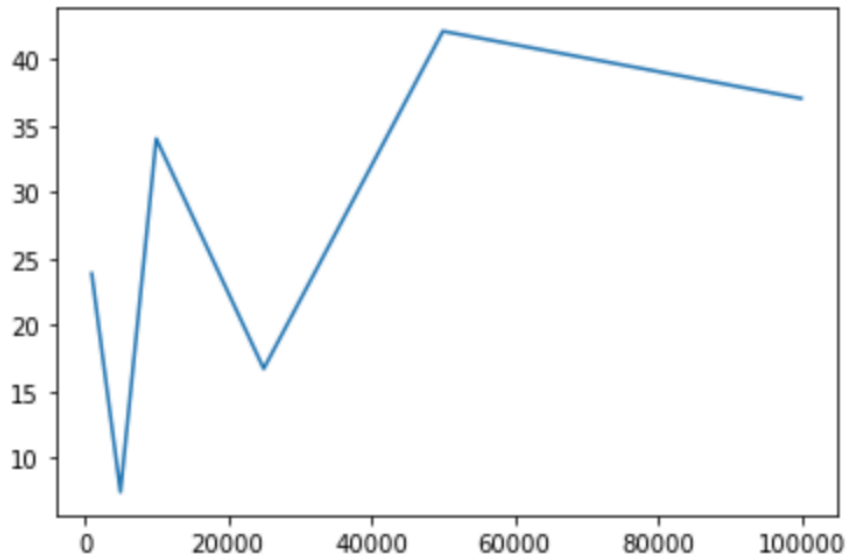
Scenario 1

When there are N active users in the network. M tweets have been communicated amongst users and their subscribers(+mentioned).
The time taken for the most famous user(maximum subscribers) to successfully tweet is plotted against total number of users communicating in the network.
M= 5000



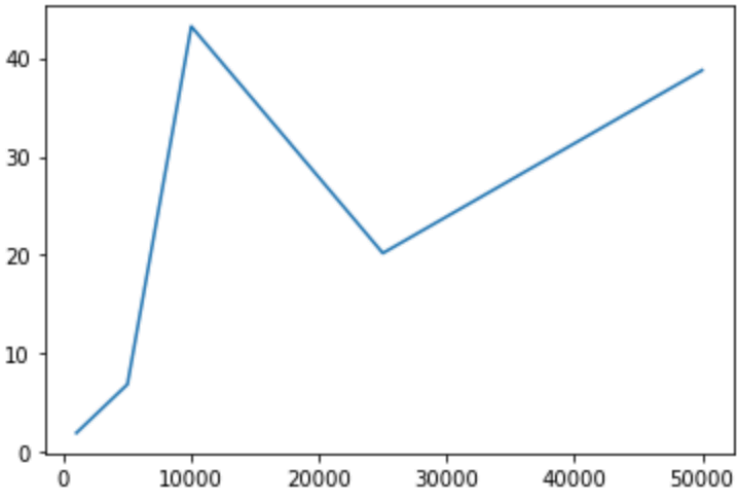
Scenario 2

When there are N active users in the network. M tweets have been communicated amongst users and their subscribers(+mentioned).
The time taken to fetch all tweets which contains Hashtag H is plotted against total number of Tweets in the network. N = 5000



Scenario 3

When there are N active users in the network. M tweets have been communicated amongst users and their subscribers(+mentioned).
The time taken to fetch all tweets which contains Mentions m is plotted against total number of Tweets in the network.
N = 5000



Output

```
"mguu":Subscribed to "obmb"
"mguu":Subscribed to "hvhn"
"mguu":Subscribed to "roua"
"mguu":Subscribed to "kzcx"
"mguu":Subscribed to "bpxw"
"mguu":Subscribed to "roua"
"kzcx":Tweet Added
"jwqs":Adding to Feed
"jwqs":Adding to Feed
"bpxw":Tweet Added
"obmb":Adding to Feed
"edvb":Adding to Feed
"kzcx":Tweet Added
"obmb":Adding to Feed
"jwqs":Adding to Feed
"kzcx":Tweet Added
"edvb":Adding to Feed
"jwqs":Adding to Feed
"jwqs":Tweet Added
"edvb":Adding to Feed
"edvb":Adding to Feed
```

```
"obmb":Adding to Feed
"roua":Adding to Feed
"edvb":Adding to Feed
"obmb":Adding to Feed
"kpwb":Adding to Feed
"roua":Adding to Feed
"kpwb":Adding to Feed
"vedf":Search results for mentions "@hvhn" are ["@hvhn","@hvhn","@hvhn",
                                                "@hvhn","@hvhn","@hvhn",
                                                "@hvhn","@hvhn","@hvhn",
                                                "@hvhn","@hvhn"]
```

Conclusion

Twitter-like engine using actor model in Erlang is successfully implemented. The architecture is partly **p2p** and partly **server-client** model(for querying and redirecting). The efficacy of this architecture has been tested with various scenarios and using **zipf** distribution. Scenario 1 has given an expected **linear growth** w.r.t the total number of users. The discrepancies in the scenario2, scenario3 is due to **invalid search** (hashtag not valid, invalid user). This architecture can be made more effective with more than one supervisor is distributed across the network. With the right number os supervisors, we can expect **logarithmic** performance for scenario 1.