Report

Karan Goel 8116-0185 Siddhesh Muley 2590-1911

Twitter

From our previous project we have genserver which is running to keep track of the user in the memory of the computer, as it is required that everything should stay on a server. Phoenix Wraps around the previous project, allowing it to work as a websocket.

We have phoenix running that keeps track of all the registered users in Twitter. Whenever a new user registers, the engine is informed and it updates the subscription table within the state of the user.

All the tweets, hashtags and mentions are also stored in the engine. If a user goes offline, all relevant tweets to that user are stored in the server and are provided to the user when they are back online.

We implemented retweet testing by randomly selecting tweets and sending them to the subscribers. These users will store the tweets IDs they have and will not keep or retweet duplicates.

Hashtags are generated as random strings preceded by a hash `#`. These hashtags can be searched by a user by querying the database of the engine. Similar functionality exists for mentions as well. Mentions, unlike hashtags are usernames preceded by the symbol `@`. These are the actors pretending to be Twitter users, we have tried to query for the same at random instance within the code.

Architecture

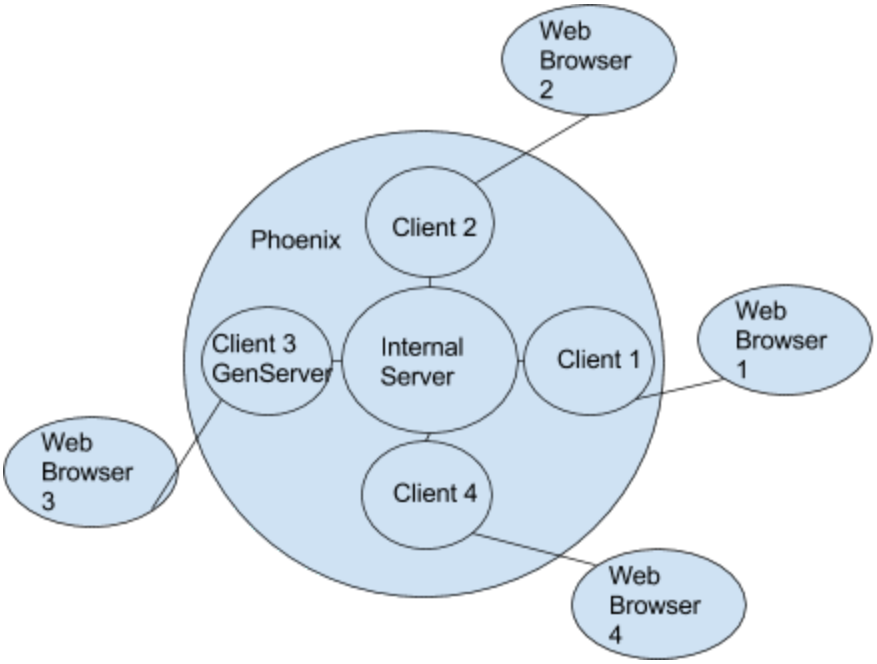

Fig : Showing Client and Server Architecture in the project

For registering we will connect to the UserSocket and it connects and then have to subscribe to the Room Channel and from room channel we will be either be getting :ok, or an :error, based on whether the user was successfully able to go to the nodes or not.

Hashtags, Mentions and Tweets have handle_in event in the room channel and hence we are able to get the value through the web sockets.

We have three main sections in the code,

1. We have server.ex which is the internal genserver which will keep all the data _DIR_*/lib/twitter_new_web/channels/server.ex*
2. We have room channel which is _DIR_*/twitter_new/lib/twitter_new_web/channels/room_channel.ex*
3. And our test Simulation in _DIR_/test/twitter_new_web/channelsroom_channel_test.ex

To run the code and get the result for the throughput we will need to follow these steps

Remote Setup:-

1. Copy the entire Folder
2. Run command mix deps.get
3. Then to run the test script for the entire code, run the *'mix test --trace test/twitter_new_web/channels/room_channel_test.exs'*
4. We can change the number of user by changing @num parameter in the same file, to do that we will use command *'vi test/twitter_new_web/channels/room_channel_test.exs'*
5. We implemented minimal UI to show that various sockets are working on web browser
6. We can run the server by 'mix phx.server'
7. Run the URL http://0.0.0.0:4000/?user=2

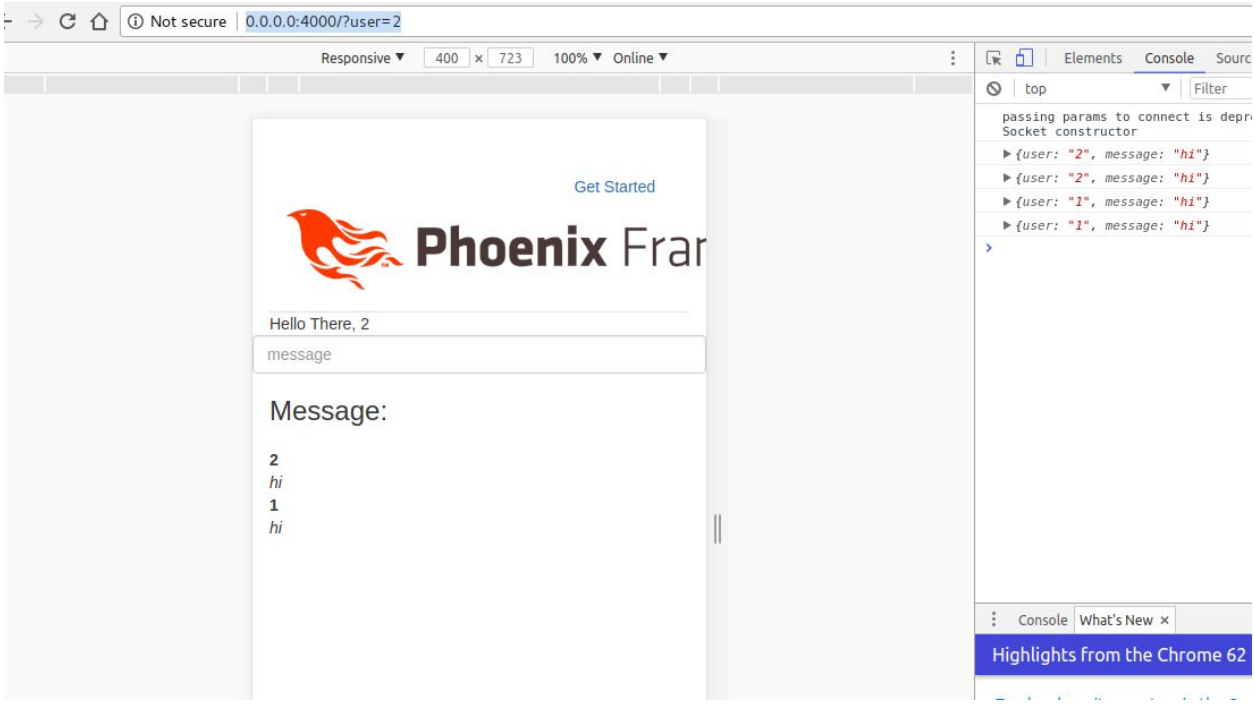

Fig: To show that sockets are working on web browser

8. Run the URL http://0.0.0.0:4000/?user=3
9. Send Message from one user to another

Zipf's law

For distribution of subscribers we have used zipf's law so some number of users will get more number of subscribers and other will have lower number of users.

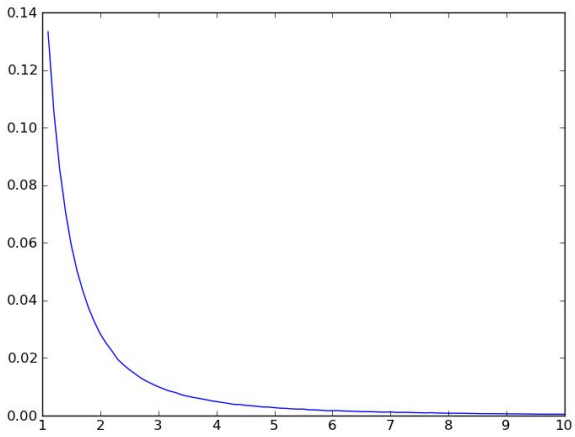

Fig : Sample Representation of zipf's distribution

$$f(k:s,N) = \frac{\frac{1}{k^s}}{\sum_{n=1}^{N}(\frac{1}{n^s})}$$

N - number of elements in distribution,

k - rank of element

s - value of exponent

Fig 1.2: this is the formula for zipf's distribution, where k is the number of user and s is an exponential constant used 1.3 in our case.

In the project part -1, we chose to randomize few of the nodes in the network to start and stop, but in this part we have decided to start and stop all the nodes in the network to emulate a worst case scenario. We distributed the subscriber according to the Ziph's distribution to be able to compare the two models and here are the results for both of codes.

Sr. No.	No. of Nodes & tweets	Throughput GenServer	*Throughput Websockets*	Difference
1.	1000	650	*613*	5.6%
2	5000	2758	*2653*	3.8%
3	10000	5486	*5226*	4.7%

Results:-

Based on our observations after the testing done on the second part of the project, we can say that while implementing sockets does provide more complex functionalities and features, there is also a significant amount of drop in the throughput of the application when compared to the non-socket implementation of the same application.

Sample Output:-

"RT: from 41 #2pOve3OJaEKXyZMkoggWwQ @32"
{"Tweets with mentions of 32",
 ["#yzsAhX7uzmGy5fLeZDIrCA @32", "#jHsw3P7qGMHwkcz9gLsWVw @32",
  "#2pOve3OJaEKXyZMkoggWwQ @32"]}
"tweet: from 98 #71JpIHaZVDkai7zHtW4o2w @69"
"tweet: from 99 #vJt6evdQJPnJO8zF3tRuoQ @95"
"tweet: from 100 #pah6jVAViW3qRAGQXWRiJA @35"
173
{"Throughput", 123}

CPU utilization

For 4 core machine

```
top - 04:08:48 up 14:21,  5 users,  load average: 2.19, 1.29, 1.09
Tasks: 301 total,   3 running, 298 sleeping,   0 stopped,   0 zombie
%Cpu(s):  0.3 us,  9.0 sy, 64.5 ni, 26.0 id,  0.0 wa,  0.0 hi,  0.1 si,  0.0 st
KiB Mem : 16343032 total,  6257700 free,  9767448 used,   317884 buff/cache
KiB Swap: 75640832 total, 73780256 free,  1860576 used.  6250980 avail Mem

  PID USER      PR  NI    VIRT    RES    SHR S  %CPU %MEM     TIME+ COMMAND
 7955 kgoel     30  10 8358300 3.437g   6688 S 478.8 22.1   1:06.25 beam.smp
 2593 akatiyar  30  10 5251592 3.811g   1564 R 100.3 24.4 398:41.54 python3.6
 3314 kgoel     20   0  157988   2000    464 R   9.3  0.0   0:18.77 sshd
 6991 kgoel     30  10 5745496 1.382g   6700 S   2.3  8.9   6:32.95 beam.smp
 7863 root      20   0       0      0      0 S   1.7  0.0   0:00.20 kworker/u16:0
 7912 root      20   0       0      0      0 S   1.7  0.0   0:00.14 kworker/u16:3
 6254 kgoel     30  10   33688   1488    928 R   0.3  0.0   0:21.77 top
```

For 8 core machine

Youtube Video Link

https://www.youtube.com/watch?v=OPSwE53BVVc&feature=youtu.be