COP 5615: Distributed Operating Systems Principles
Project 3
Chord based Peer 2 Peer system implementation based on Erlang Programming Language

Team members

* Rahul Vikram Porwal UFID: 44590947 Email: rahulporwal@ufl.edu

* Pujan Narendra Kothari UFID: 70612594 Email: p.kothari@ufl.edu

Introduction

In a traditional Client-server model, There is only 1 server and multiple clients. It becomes very slow and overloading for the server to serve the needs of all the clients at the same time. Clients also receive the information very slowly as 1 server is being shared amongst every other client. On the other hand, We have the Peer 2 Peer Model. It has a very unorthodox approach, where the client can also act as server and therefore there are multiple clients acting as clients. They can fulfil a lot of requests very fast and load is light on each client.

Instructions to execute the project

1) Install the latest version of erlang.
2) Download the **project3.zip** and unzip it to any desired path.
3) Run command prompt at the file location with commands - erl
   To run erlang environment
4) Compile the erlang file-
   c(project3.zip)
5) Then, To execute the Chord protocol using p2p system, Follow the command:
   project3_chord:start_network(<number of nodes>,<number of requests per node>).
   Eg. project3_chord:start_network(10,5)
   Where,
   10 is the number of nodes, 5 is the number of requests per node.
6) It will print the average hop count at the end.

Note- Before every execution, Run flush(). Command in the erlang environment to empty all registered processes.

What is working

We were able to implement a scalable key location based lookup scheme on chord protocol for a peer 2 peer system.

We are able to generate the number of nodes that are given as input, joining the nodes in the chord ring. We are also able to create the finger tables for each node (with m entries) and get the IDs for active nodes present for ring positions NodeID+2^0 to NodeID+2^(m-1), where m is the number of bits based on which the ring size is determined (2^m). The nodes then generate requests based on the number of requests and we get the keys to be looked up. We have also performed the scalable key location based lookup scheme and route the key to the required node in O(logN). We have calculated the average hop count for the key to be routed to the required node.

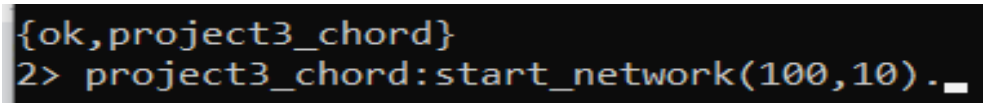
Output:-

- **Number of Nodes- 100, Number of requests per node - 10**

  Finger table Map Refers to the routing table of each node.

  Look up key indicates the key being looked up.

  Node Id reached indicates the NodeID to which the key has been routed



```
6> Finger Table Map is #{0 => 634959,1 => 634959,2
                        4 => 634959,5 => 634959,6 =>
                        8 => 634959,9 => 634959,10 =>
                        12 => 634959,13 => 634959,14
                        16 => 634959,17 => 634959,18
                        20 => 1591706}
6> Lookup Key is: 301822
6>
6> Avg Hop Count is : 5.83
```

- **Number of Nodes- 50, Number of requests per node - 2**

```
Eshell V13.0.4  (abort with ^G)
1> project3_chord:start_network(50,2).
```

```
4> Finger Table Map is #{0 => 777929,1 => 777929,2
                        4 => 777929,5 => 777929,6 =>
                        8 => 777929,9 => 777929,10 =
                        12 => 777929,13 => 777929,14
                        16 => 777929,17 => 777929,18
                        20 => 1842250}
4> Lookup Key is: 119986
4>
4> Avg Hop Count is : 4.37
```

No. of Nodes	Average Hop Count
10	2.38
100	5.83
250	7.02
500	8.24
1000	9.78

What is the largest network you managed to deal with

- Largest network we were able to implement was of

  Number of nodes - 1000, Number of requests per node - 10

  Node server list indicates list of all active nodes in the chord ring.

  Average hop count indicates average hops taken for each key to be routed to the required node.