COP5615_Chord Protocol_Project

Team Members

- Rishabh Srivastav: UFID 7659-9488
- Ashish Sunny Abraham: UFID 6388-7782

Outline of the project

This project aims to design the Chord protocol and a simplistic object access service to demonstrate the Chord protocol's utility using Erlang and Actor Model.

Commands to run the program

- Compile all the .erl files in the zip folder (main, nodeCreation, stabilizeChordRing, chordRing, utilityFunctions).
- Run the project by running the command “main:start().” in the erl shell.
- Enter the number of nodes, message requests and failure nodes, for example, 1000,10,0 (Refer to bonus report for failure node implementation).

Implementation Details

Chord Protocol:

- In our implementation of the chord protocol, we ask the user to enter the number of nodes ‘n’, based on which we create 2^m nodes (M) where m = ceiling of log2n.
- We then select n nodes from 1 to M to participate in the algorithm. The remaining M-n nodes are missing and their information will be stored with their successor.
- Following, a random node is selected which needs to be searched by every other member in the chord ring exactly the number of times as per the message request parameter given by the user in the input.
- We then create finger tables for each created node and perform stabilization upon them to calculate the average number of hops required for one particular node to search for another node.

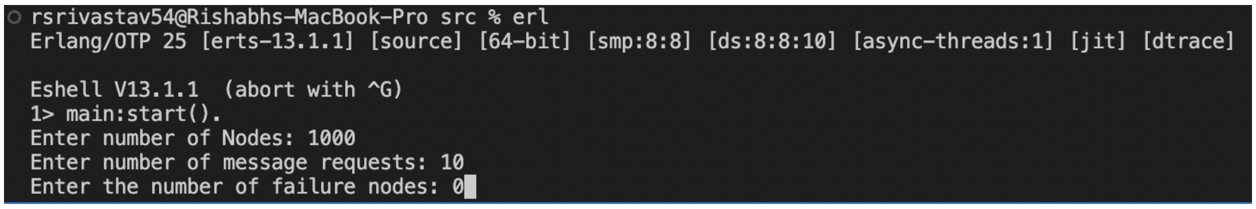
What is working?

- The chord protocol is implemented using methods mentioned in the research paper provided in the problem statement. The model is working as expected.
- There were several conflicts when creating distinct node IDs as a result of the hashing procedure as specified in the criteria proceeded by m bit reductions. In order to overcome this, the project incorporates a random number approach.
- The largest network tested with is 2000 nodes with 10 messages each, beyond which the program starts throwing multiple deadlock errors. The average number of hops required for a message to reach its targeted node was in the range of [2.6, 4.5] when tested from 100-2000 nodes. The value came out to be 3.45 hops when it was averaged across 20 runs. Hence, on an average it takes 3.45 hops for a node to search any other node in the chord protocol meaning the look-up time taken is in the order of log n.

Observations:

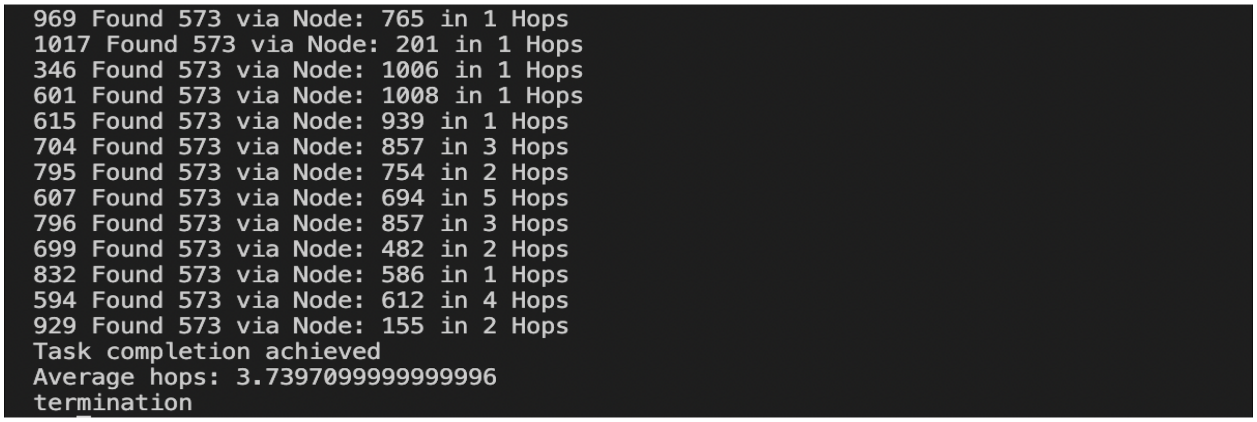
Number of Nodes	Average Hops
100	2.6
500	3.2
1000	3.7
2000	4.5

Running program for 1000 nodes, 10 message requests, 0 failure nodes:



```
rsrivastav54@Rishabhs-MacBook-Pro src % erl
Erlang/OTP 25 [erts-13.1.1] [source] [64-bit] [smp:8:8] [ds:8:8:10] [async-threads:1] [jit] [dtrace]

Eshell V13.1.1  (abort with ^G)
1> main:start().
Enter number of Nodes: 1000
Enter number of message requests: 10
Enter the number of failure nodes: 0
```



```
969 Found 573 via Node: 765 in 1 Hops
1017 Found 573 via Node: 201 in 1 Hops
346 Found 573 via Node: 1006 in 1 Hops
601 Found 573 via Node: 1008 in 1 Hops
615 Found 573 via Node: 939 in 1 Hops
704 Found 573 via Node: 857 in 3 Hops
795 Found 573 via Node: 754 in 2 Hops
607 Found 573 via Node: 694 in 5 Hops
796 Found 573 via Node: 857 in 3 Hops
699 Found 573 via Node: 482 in 2 Hops
832 Found 573 via Node: 586 in 1 Hops
594 Found 573 via Node: 612 in 4 Hops
929 Found 573 via Node: 155 in 2 Hops
Task completion achieved
Average hops: 3.7397099999999996
termination
```