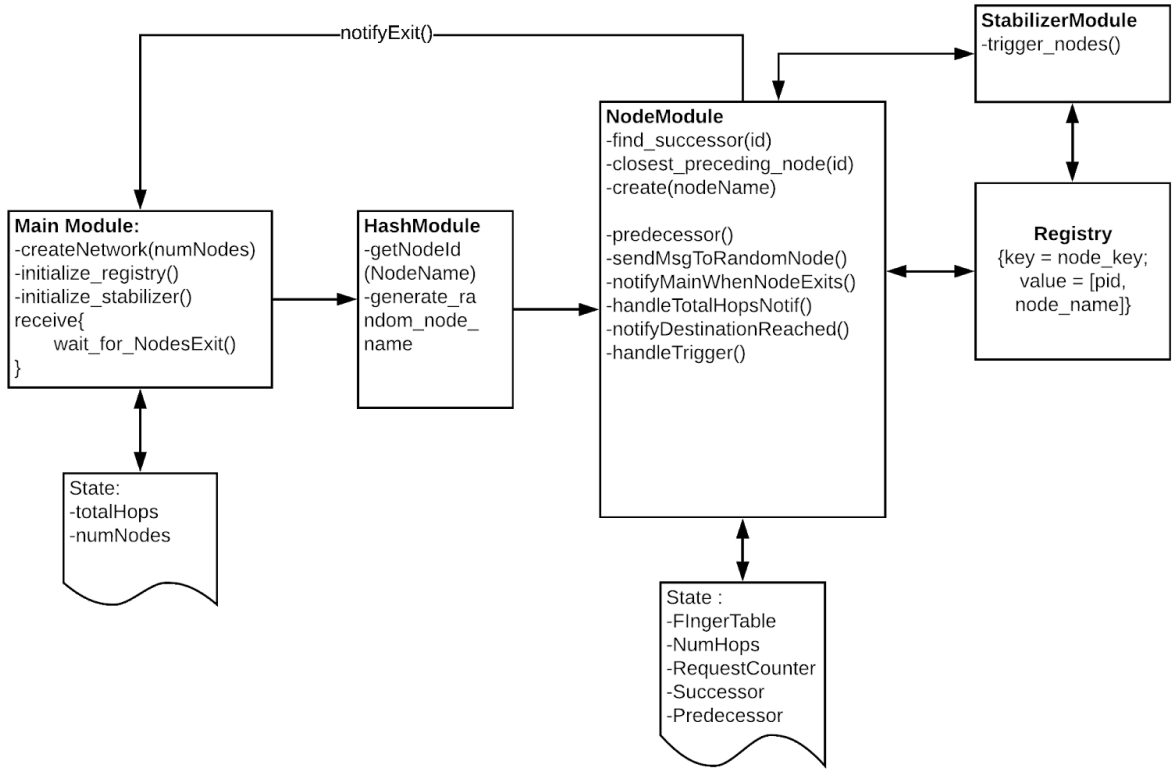
COP5615: Distributed Operating Systems (Fall 2018)
Project 3 - Chord Protocol

Submitted by:
Kunwardeep Singh (UFID: 2421-3955)
Gayatri Behera (UFID: 3258-9909)

Chord Algorithm Implementation:



In this project we implemented the Chord protocol for message passing in peer-to-peer applications. The chord protocol aims to achieve efficient location of data. It enables the same with the use of a distributed hash table, storing the key-value pairs and it's retrieval. The node ids and keys are generated by means of the hashing algorithm SHA-1. The respective nodes that are part of the network are traversed in a clockwise manner while determining the next node that has to be accessed. The sequence of next nodes to be accessed is maintained in a finger table. The maximum number of node addresses that the finger table has references to is determined from the value 'm'. Value of 'm' is utilised for determining the hash values of the nodes.

Chord protocol mandates the implementation of the following API:

1. create()
2. join(node)
3. stabilize()
4. notify(node)
5. fix_fingers()
6. check_predecessor()

On initialisation of the network, the nodes are required to send message to a random lookup key which can vary from 0 to 2^m. The traversal of any 'request' needs to happen in a clockwise direction and the number of 'hops' taken is accounted for. This is done by keeping track of the current hop count for a particular request. Depending on the number of requests originally input from the console, each node sends that many requests. Each request seeks out a node arbitrarily chosen to reach. It references each node's finger table for the destination node address or an intermediary node to transmit to. This transition from one node to another is called a 'hop' and the number of hops from initialization node to destination node is noted. The average number of hops made is computed taking into account the total number of hops made, the number of nodes and the number of requests. Once any node has met the criteria for number of requests it's terminated.

The main process waits until all nodes are terminated. Then it prints the required average number of hops using the formula:

$$\textit{Average hops} = \frac{Total\ Number\ of\ hops}{Number\ of\ Nodes \times Number\ of\ Requests\ per\ node}$$

Bonus Section:

For implementing this section - number of nodes that were to be terminated were decided on from the beginning. These many nodes are selected at random and are forcefully terminated. Meanwhile, stabilize() and fix_fingers() method are being called continuously to maintain the integrity of the chord network.

Running the program:

To start the program, use:

```
mix run proj3.exs [num_of_Nodes] [num_of_Requests]
```

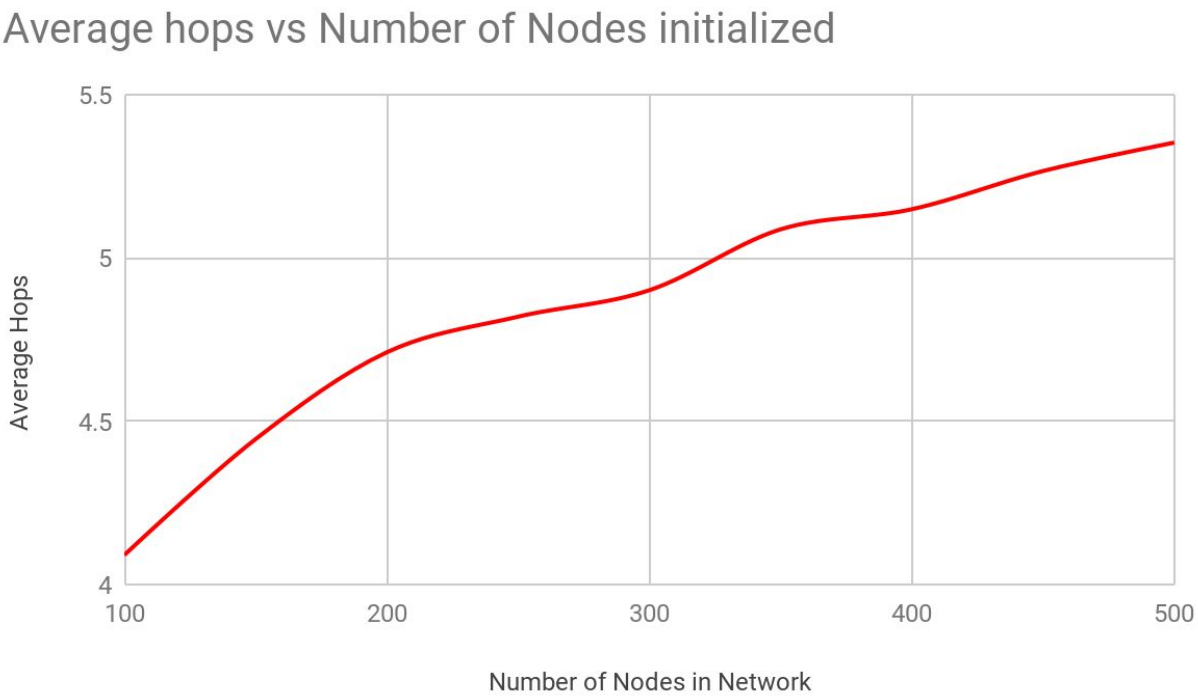
For bonus section implementation:

```
mix run proj3.exs [num_of_Nodes] [num_of_Requests] [num_of_Nodes_to_fail]
```

Observations:

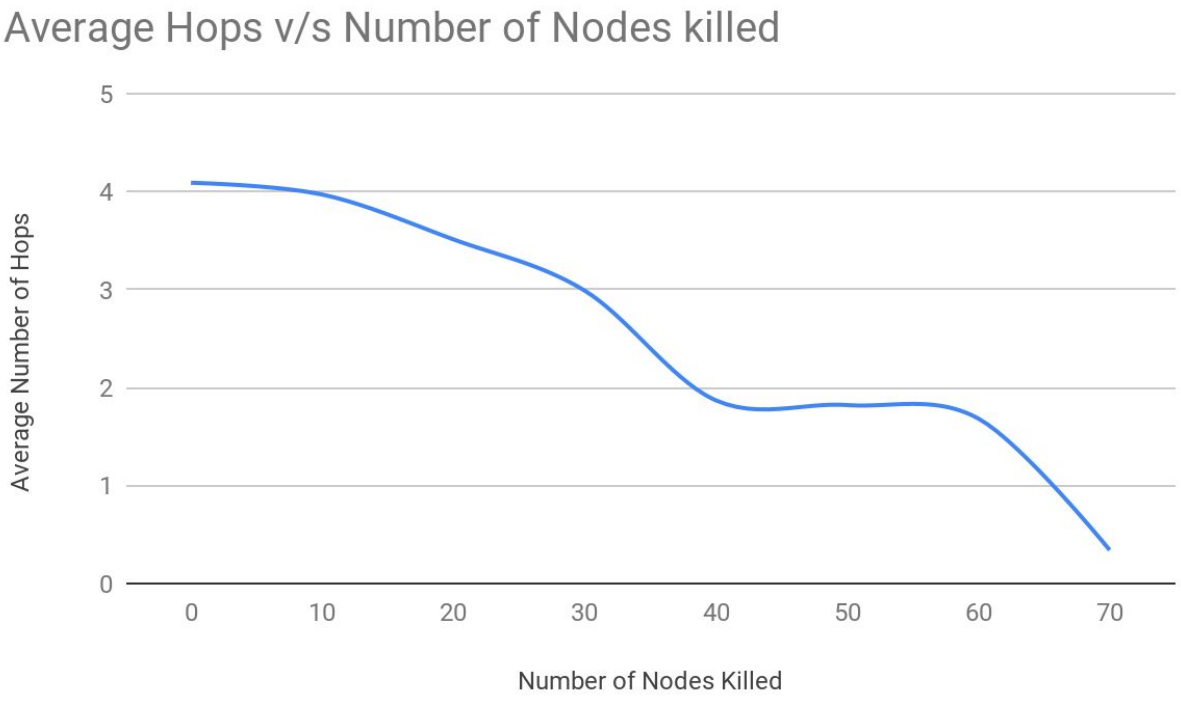
a. Regular run:
   Number of Requests: 10
   Initial number of Nodes in Network: 100

   On running the program after varying the number of nodes we observed that as the number of nodes set during initialisation were increased, the average hop count hiked.



b. On failing nodes:
Number of Requests: 10
Number of Nodes in Network: 100

On running the program after varying the number of failure nodes we observed that the fewer the number of nodes left in the network, brought about by failing a select number of nodes during iteration, resulted in decrease of the number of hops. Also, the network remained resilient as the stabilization logic was working which kept on updating the neighbours and the finger table for each node. Without the stabilizer, there is an increased probability of deadlocks and instability of the network. But as the stabilized network didn't crash, it proves that it is resilient.



References:

1. https://en.wikipedia.org/wiki/Chord_(peer-to-peer)
2. https://pdos.csail.mit.edu/papers/ton:chord/paper-ton.pdf