PROJECT 3: CHORD P2P SYSTEM AND SIMULATION

HOW TO RUN THE PROGRAM

- Save the file on your system as project3.erl
- The latest version of Erlang/OTP 25.1 is recommended.
- Run the erl command on your system to ensure it is installed correctly.
- Move to the directory where the program is saved.
- In the terminal run the command **c(project3).**
- In the terminal run the command
  project3:start(Number_of_Nodes, Number_of_Requests).
  *****Note: Execution for each given in the demo video.
  *Samples: project3:start(10,10).
- The terminal gives output.
  The First node with it's hashed value.
  The number of the node that was created.
  The map associating the node with the process id.
  The program then sets the successor and predecessor according to the assignment mentioned in the paper
  The map of the same.
  The Fingertable for all these pids.
  The average number of hops.
  * These get updated according to chord protocol.

READMEREQUIREMENTS AND ANALYSIS

-
- What is working
  The join, stabilize, and other APIs stated in the research article are used to create the chord protocol. The implementation is consistent according to the work of the research paper.

- What is the largest network you managed to deal with?
  The maximum number of nodes that may be tested is 2000, although stabilizing so many nodes takes a lot longer. The software performs better with more messages. The system reached its limit with a chord that can include up to 2000 nodes with 10 messages each and had a runtime of 37 minutes[Approx]. It took hops approximately in the range of 4-7 for a message to arrive at its desired destination. The result was a mean of 5.697 hops when the value was calculated for 10 consecutive runs with a median of 1.73. This means that the chord protocol requires approximately an average of 5.697 hops for each node to look for any desired key it is looking for. This demonstrates that a lookup process takes the order of log n time as mentioned in the requirements.

Description
Actor Model
The actors are spawned in the 64th line of the code and after that work is assigned to them by the server.

Input
Number_of_Nodes - Number of nodes to be created in the peer-to-peer network
Number_of_Requests - Number of nodes each node must make to the other nodes/peers in the network.

Description
- By utilizing the join and routing techniques as described in the paper, the Chord protocol for distributed systems was executed. This step is followed by, creating a network ring comprising of the nodes/peers and a finger table for each node that is utilized for storing the routing information for approximate log(n) peers present in the network where n is the total number of nodes in the network, we were able to build the network topology.
- The chord protocol algorithm initiates by creating nodes equal to Number_of_Nodes, giving each node a Node ID that has been subjected to SHA-1 hashing, and generating requests for each node in the peer network equal to the number of requests entered by the client/user. The program is then utilized for calculating the average number of hops required for each peer/node in the network to locate the desired value keys, which are likewise SHA-1 hashed as mentioned in the specifications.

Implementation of the Chord - P2P System
- The program is initiated by taking the master/server actor and building a structure like a chord ring for actors acting as network peers. For each node, we define a predecessor peer node in the network and a successor peer node in the network. Every node maintains a finger table that keeps entries for when it resolves keys to nodes.
- SHA – 1 algorithm is utilized for hashing the Node IDs and the keys in the network as required.
- The network when set up completely and ready for operations, each node is given a set of fixed requests to complete, such as key lookups defined in the function. During this phase of execution, the finger table entry's nodes are recursively called by a lookup function, increasing the hop count with each subsequent call, and stored in a global variable. The total number of hops divided by the total number of requests yields the average hop count.
- The program terminates with each peer node reporting to the master/server node when it has processed all the requests assigned to it.

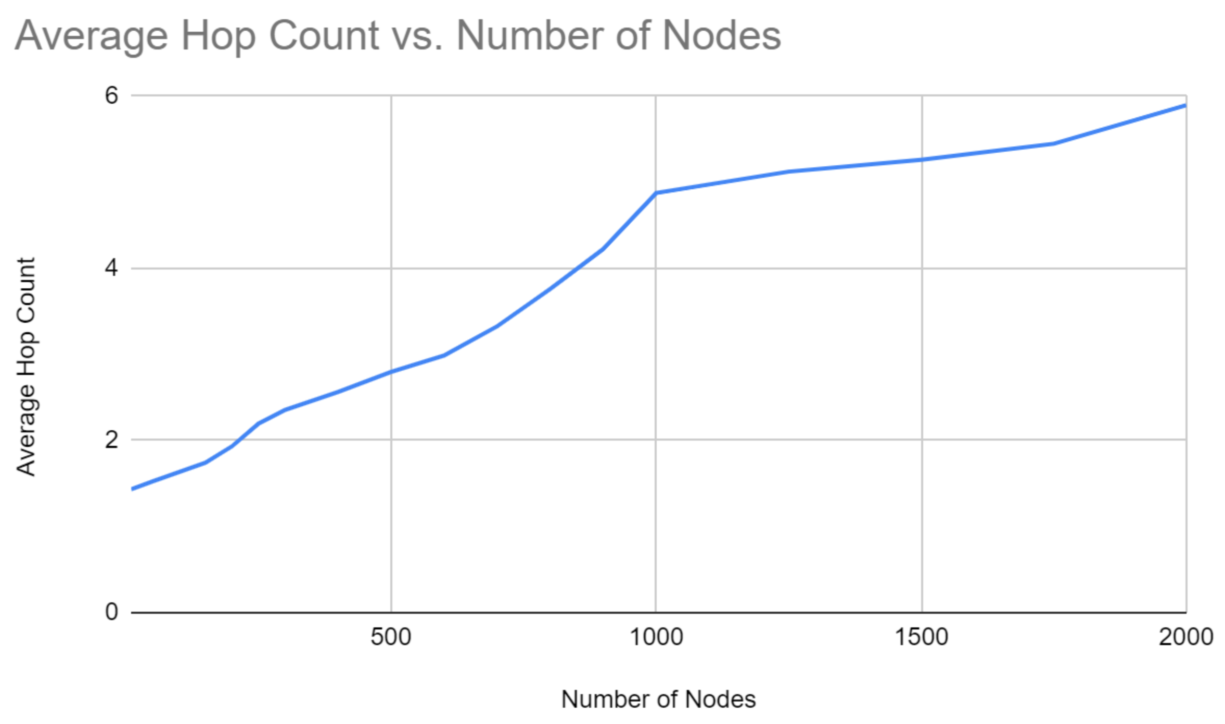
Termination of the Program
When all actors/nodes present in the peer network ring have successfully generated the Number_of_Requests that the user set using command line input, the program terminates giving all

the information in the order as specified under the HOW TO RUN THE PROGRAM section. A request is created and sent by each peer node once per second, and each peer node notifies the master/server node when it has completed execution and successfully processed all of the request messages allotted to it.

Results
According to Number of Nodes

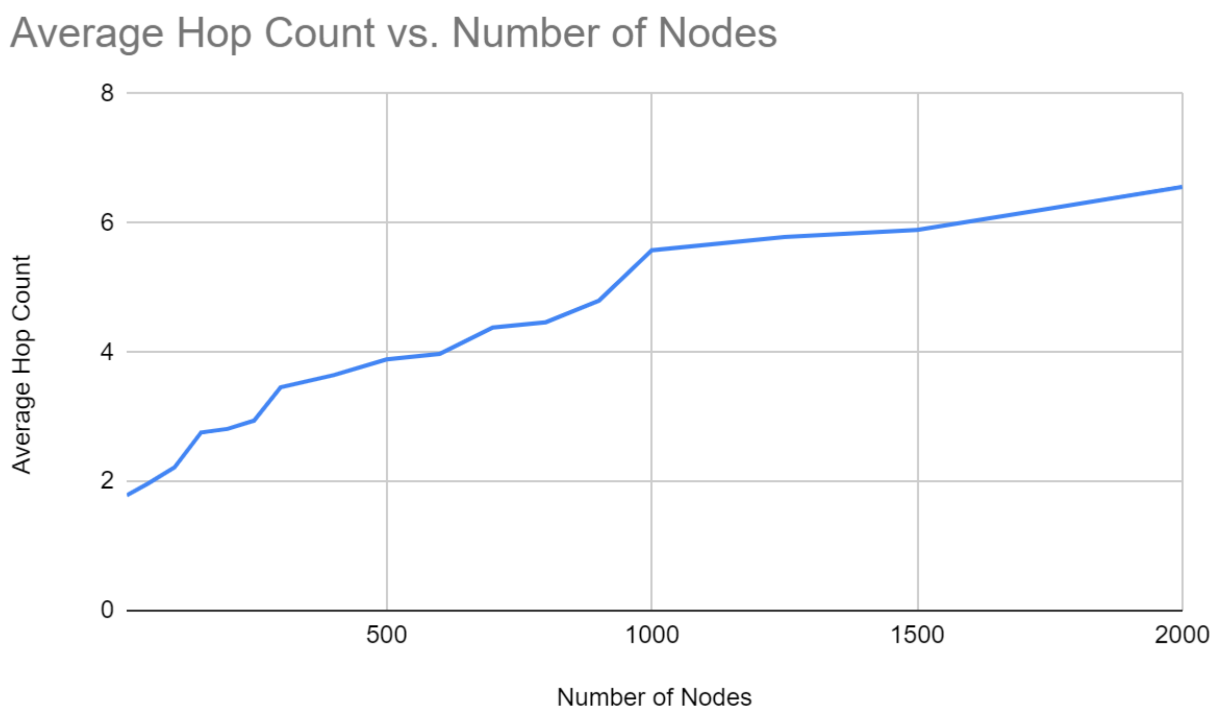
Number of Nodes	**Number of Requests per Nodes**	**Average Hop Count**
10	10	1.432
50	10	1.524
100	10	1.633
150	10	1.743
200	10	1.932
250	10	2.197
300	10	2.356
400	10	2.563
500	10	2.798
600	10	2.987
700	10	3.327
800	10	3.759
900	10	4.223
1000	10	4.876
1250	10	5.124
1500	10	5.263
1750	10	5.348

2000	10	5.697



Number of Nodes	**Number of Requests per Nodes**	**Average Hop Count**
10	5	1.785
50	5	1.964
100	5	2.218
150	5	2.756
200	5	2.812
250	5	2.939
300	5	3.457
400	5	3.645

500	5	3.889
600	5	3.974
700	5	4.383
800	5	4.463
900	5	4.797
1000	5	5.579
1250	5	5.783
1500	5	5.893
1750	5	6.128
2000	5	6.559



Observations

- As number of Nodes increase there is a steady rise in the average hop count.
- As number of requests decreases the average hop count for the same number of nodes increases.
- The curve begins to flatten a little as the number of nodes crosses 1000.