DISTRIBUTED OPERATING SYSTEMS PRINCIPLES

COP5615 - REPORT

PROGRAMMING ASSIGNMENT - 2

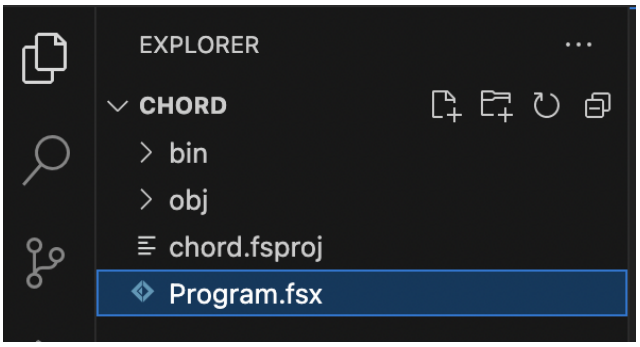
PA2_Team 29		
NAME	UFID	EMAIL
Nitesh Varanasi	38666016	varanasin@ufl.edu
Raghav rathi	98114622	raghavrathi@ufl.edu
Sri Charan Pabbathi	80984724	spabbathi@ufll.edu
Sreekar Reddy Nathi	28243129	sreekarreddnathi@ufl.edu

CODE EXECUTION

Execution of Chord p2p protocol Needed some requirements.
Requirements:

1. DOTNET SDK
2. F# Ionide extension for VSCode

Following is the program directory where the implementation is present in Program.fsx file



To compile and run the Program.fsx:

1. Start a command prompt or terminal.
2. Go to the directory where the Program.fsx is located.
3. Run the command >> **dotnet fsi Program.fsx <num_of_nodes> <num_of_requests>**

Here,
numNodes - This is an integer that represents the number of peers within the Chord network.

numRequests - This refers to the number of requests that each peer is responsible for handling.
Ex: dotnet fsi Program.fsx 50 5

If Step 3 does not execute, try Step 4 and Step 5 and then run Step3
Step 4: dotnet add package Akka -version 1.4.25
Step 5: dotnet add package Akka.Fsharp -version 1.4.25

What is working?

Creation of Peer to Peer Chord Network Ring :
Successfully created network ring with a certain number of nodes (numNodes) with respect to the requirements mentioned in the paper. Each node has an integer value (identifier) associated with it.

Our project is based on the Actor model of computation using the Akka framework, which orchestrates concurrent operations by employing a multitude of actors that communicate through message-passing and also maintains the state of variables like fingertable, successors, etc. The primary essence of the project is to create a Chord ring—a decentralized peer-to-peer system, and perform key lookups within it, measuring efficiency and hop counts.

Here are some key functionalities:

- **Network Creation and Node Initialization**: The system starts by creating the Chord network. If it's the very first node, it initializes the network. Otherwise, it joins the existing Chord network. Every node has a unique identifier, and this identifier determines its position within the ring.

- **Actor System and Message Handling**: The implementation uses the Akka Actor framework. Each node in the Chord network is represented as an actor, and these actors communicate through message-passing. This means that nodes can send, receive, and process messages to execute various Chord operations, such as Join, FindSuccessor, and Stabilize, etc.

- **Dynamic Finger Table**: Each node maintains a finger table, which is essential for the quick lookup of keys. This finger table gets periodically updated and fixed using the FixFingers operation, ensuring efficient routing and key lookups.

- **Key Lookup Mechanism**: The scalable key lookup is achieved through the ‘LookupKey’ operation. A node can query a specific key, and the system will navigate through the nodes using the finger tables, counting the number of hops taken to reach the node responsible for that key. This is essential to gauge the efficiency of the network.

- **Node Stabilization**: Periodic stabilization is implemented to make sure the network remains consistent as nodes join and leave. The ‘Stabilize’ operation checks for newly joined nodes and updates the successors and predecessors accordingly.

- **Performance Metrics**: After simulating a series of lookup requests, the system calculates and outputs the average number of hops required to find a key. This metric is crucial for understanding the efficiency and scalability of the implemented Chord protocol.

- **Finger Table Display**: For debugging and verification, the system can display the finger table of a particular node using the ShowFingerTable operation. It showcases the immediate successors that a node would consider when routing a request or a key search.

The chord network is initiated with the creation of the first node. As the network evolves, additional nodes are introduced, with each node defined by its id, finger, successor, and predecessor. These nodes inherently comprise finger tables, which consist of entries detailing potential nodes to skip to during key resolution.
Nodes are then allocated specific lookup requests. The key lookup mechanism involves recurrent calls to the lookup function on nodes specified in the finger table. With each iteration, there's an incremental increase in the hop count. Finally the average number of hops (avgHops) is calculated by dividing the totalHops (which is the cumulative sum of hops for all requests across all nodes) by the product of numRequests (number of requests) and numNodes (number of nodes).

In essence, this code provides a practical implementation of the Chord protocol, which is used for organizing and querying keys within a distributed network. It sheds light on the dynamics of Chord networks, emphasizing their evolution, stability, and capability to swiftly pinpoint keys with minimal hops. The results of the experiments demonstrate the protocol's effectiveness in practice.

OUTPUTS

Outputs For Number of Nodes (5,10,20,40,100) and Request counts 5

```
vnitesh@Varanasis-MacBook-Air chord_program % dotnet fsi Program.fsx 5 5
Num Nodes: 5
Num Requests: 5
maxKeys is 1048576
total hops is 48
Average number of hops: 1.920000
vnitesh@Varanasis-MacBook-Air chord_program % dotnet fsi Program.fsx 10 5
Num Nodes: 10
Num Requests: 5
maxKeys is 1048576
total hops is 120
Average number of hops: 2.400000
vnitesh@Varanasis-MacBook-Air chord_program % dotnet fsi Program.fsx 20 5
Num Nodes: 20
Num Requests: 5
maxKeys is 1048576
total hops is 277
Average number of hops: 2.770000
vnitesh@Varanasis-MacBook-Air chord_program % dotnet fsi Program.fsx 40 5
Num Nodes: 40
Num Requests: 5
maxKeys is 1048576
total hops is 670
Average number of hops: 3.350000
vnitesh@Varanasis-MacBook-Air chord_program % dotnet fsi Program.fsx 100 5
Num Nodes: 100
Num Requests: 5
maxKeys is 1048576
total hops is 1999
Average number of hops: 3.998000
```

Outputs For Number of Nodes (250,500,700,1000) and Request counts 5

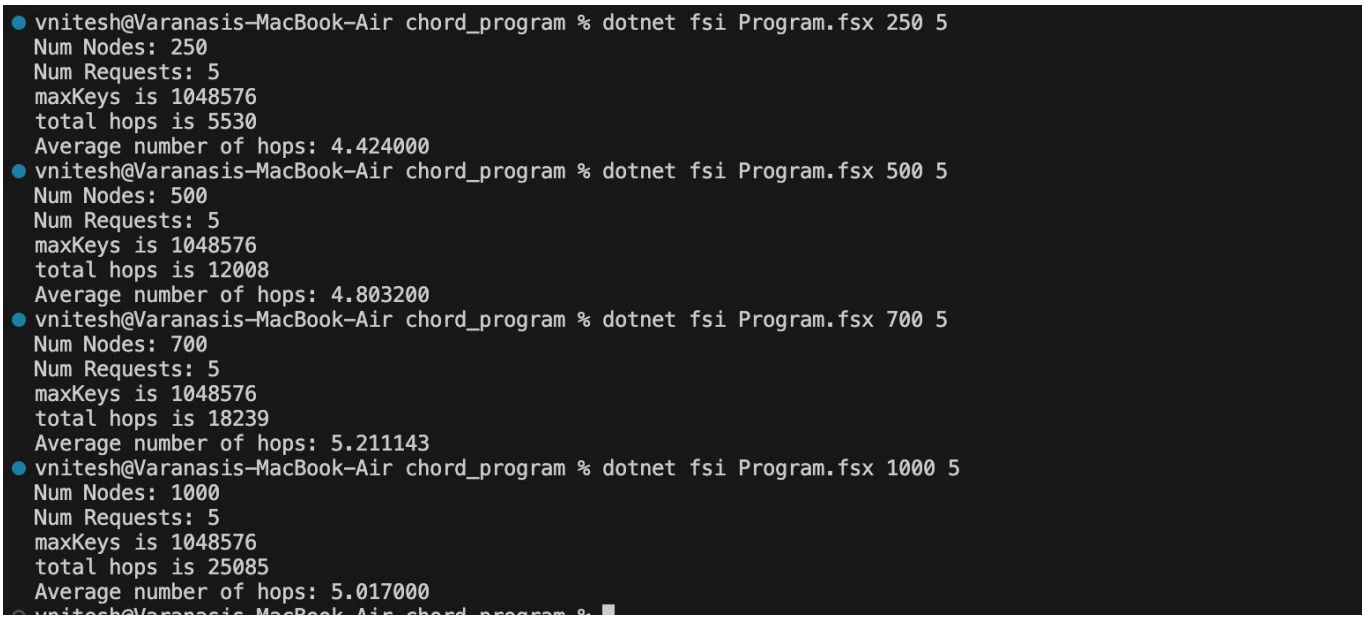
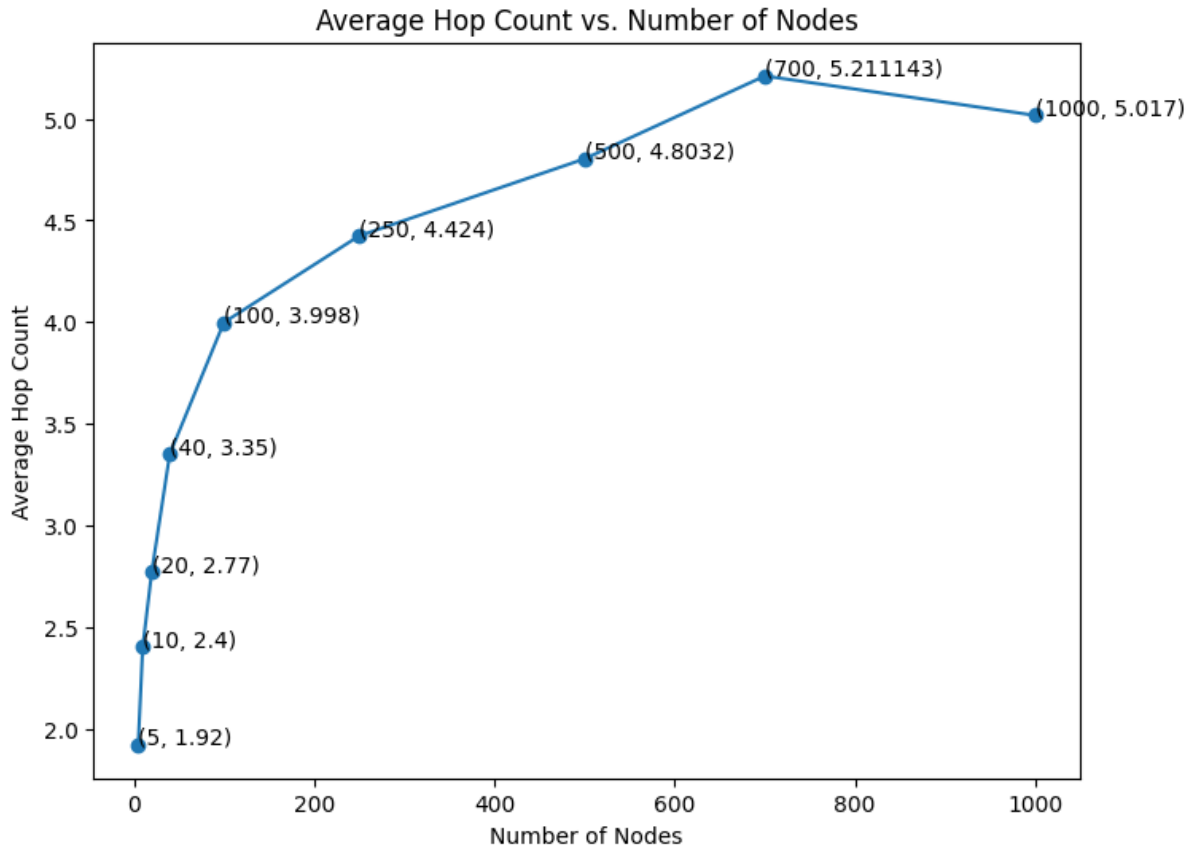


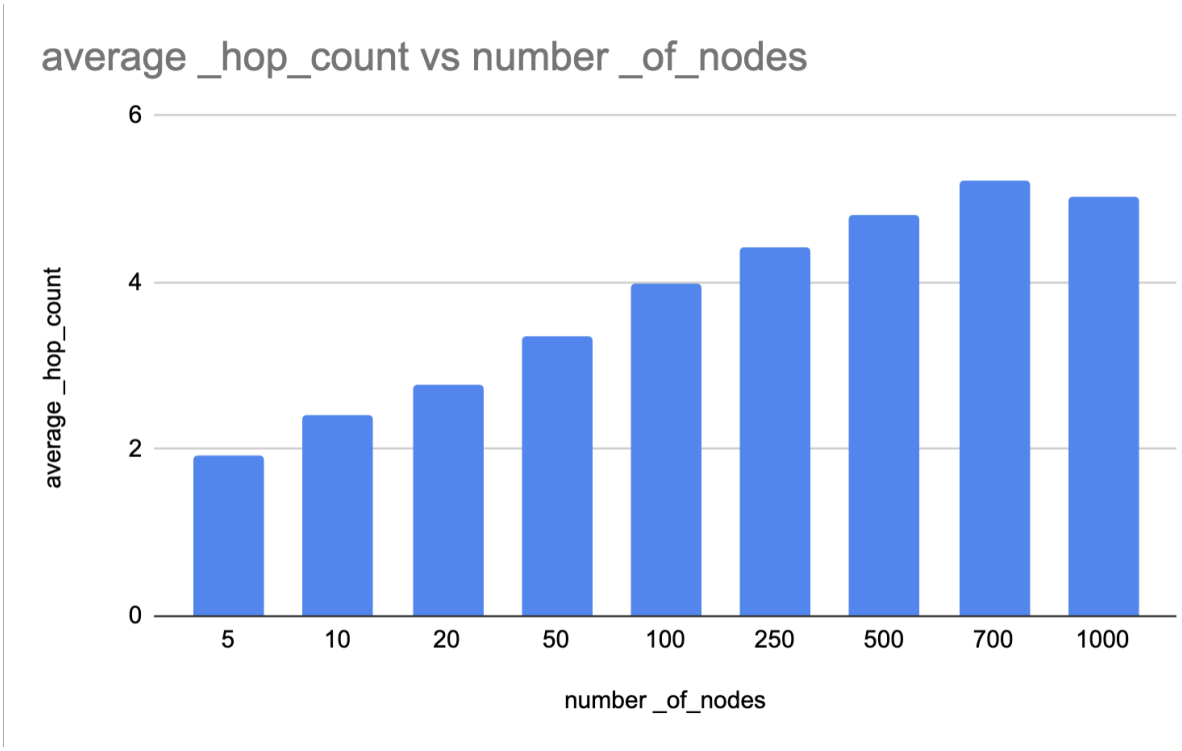
Table consisting of No. of nodes, Avg no. of Hops and Total number of Hops

Number of nodes	Average hop count	Total number of hops	Number of Requests
5	1.920000	48	5
10	2.400000	120	5
20	2.770000	277	5
40	3.350000	670	5
100	3.998000	1999	5
250	4.424000	5538	5
500	4.803200	12008	5
700	5.211143	18239	5
1000	5.017000	25085	5

Graphical Representation:

Graph for “number of nodes” VS “average hop count” Program Output





ASSUMPTIONS ABOUT THE PROTOCOL

- **Finger Table Initialization**: Nodes initialize their finger tables with a default successor. This method expedites the joining process. However, in an authentic Chord system, the finger tables typically evolve dynamically as nodes become more acquainted with the network.

- **Static Configuration:** The identifier space size is statically set (m=20). Although this simplifies the implementation, real-world applications might necessitate a more flexible approach where such parameters can be adjusted to align with the network's size and attributes.

- **Network Resilience:** The current implementation lacks explicit mechanisms to handle unexpected node departures or failures. An operational Chord network would require robust fault tolerance and recovery solutions to counteract such challenges.

- **Periodic Stabilization:** The system carries out stabilization routines at regular intervals to align with the Chord protocol's requirements. It presumes a consistent interval between stabilizations, which might need tweaking in fluctuating network conditions.

  Scalability Considerations: The script provides a foundation for implementing the Chord protocol without explicitly setting predefined constraints for node count or request numbers. However, for practical simulations, scalability considerations might be necessary to represent diverse, large-scale network scenarios effectively.

  In conclusion, while the code effectively conveys the core mechanisms of the Chord protocol, it integrates certain assumptions regarding network reliability, synchronization, fault tolerance, and system dynamics. These assumptions might need reevaluation and adaptation to better suit a real-world, extensive, and ever-evolving P2P ecosystem.

What is the largest network you managed to deal with?

We tested the Chord protocol code with network sizes from 5 to 1,000 nodes. For every node, we made it look up 5 keys. The code keeps track of the number of steps (hops) it takes to find each key. The results showed that as the network size increases, the number of hops grows in a predictable, slow manner.

The results highlighted that as the network size grew, the increase in hops remained slow and predictable. This demonstrates the inherent efficiency of the Chord protocol and its ability to handle larger networks without significantly increasing the time required for key lookups.

SUMMARY

This code provides a practical implementation of the Chord protocol, which is used for organizing and querying keys within a distributed network. It sheds light on the dynamics of Chord networks, emphasizing their evolution, stability, and capability to swiftly pinpoint keys with minimal hops. The results of the experiments demonstrate the protocol's effectiveness in practice.