COP5615- Distributed Operating System Principles Fall 2023 Programming Assignment - 2

REPORT

TEAM - 26

NAME	UFID	EMAIL
VIJAY ABHINAV TELUKUNTA	86262606	vtelukunta@ufl.edu
HARSHITH MUNDADA	70221700	harshith.mundada@ufl.edu
INDU POTLA	84964176	potla.in@ufl.edu
SUCHITHRA MACHA	19116100	smacha@ufl.edu

CONTENTS

How to run the program.

Move to the project directory (Chord) and run the following command:

dotnet run <number of nodes> <number of requests>

Ex : dotnet run 50 10

Introduction

The program code is written in F# and uses the Akka.NET framework to create a distributed Chord protocol implementation. Here's an explanation of how the program is running:

1. It starts by including the necessary libraries using '#r' directives. The program uses the Akka.NET framework, and these directives import the required packages.

2. The 'MainMessageTypes' and 'ChordMessageTypes' define the message types used in the program.

3. It initializes an Akka.NET ActorSystem named "Chord."

4. The 'Chord' function defines an Akka.NET actor that implements the Chord protocol. The actor handles various messages, such as initialization, node creation, joining, stabilizing, and finger table maintenance.

5. Inside the 'Main' function, the program initializes and configures a set of Chord nodes. It randomly generates node IDs and sets up a Chord ring. Each node is an Akka.NET actor representing a Chord node. It creates a Chord ring and performs stabilization and other Chord operations.

6. The 'loop' function inside the 'Main' actor continuously listens for messages and processes them based on their type. The initialization message 'InitializeConfiguration' is used to set up the Chord ring and start the nodes. It generates random node IDs, ensures that they are unique, and establishes the Chord ring by having nodes join one another. Stabilization processes are scheduled for each node, and when all nodes have been set up, the program calculates and prints the average number of hops for lookups.

7. The 'mainRef' factor is spawned using the 'spawn' function, representing the entry point of the program. It starts by sending an 'InitializeConfiguration' message to set up the Chord ring.

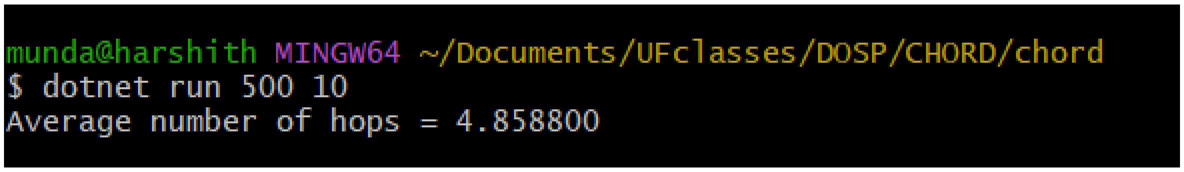
8. Finally, the program enters a wait state using ‘Console.ReadLine()’ to keep the actors running. This is a common approach in Akka.NET to ensure that the actors continue processing messages and don't terminate prematurely.

In summary, the program creates a distributed Chord ring using Akka.NET actors and simulates Chord protocol operations like node joining, stabilization, and finger table maintenance. It calculates and prints the average number of hops for lookup operations in the Chord ring when all operations are completed.

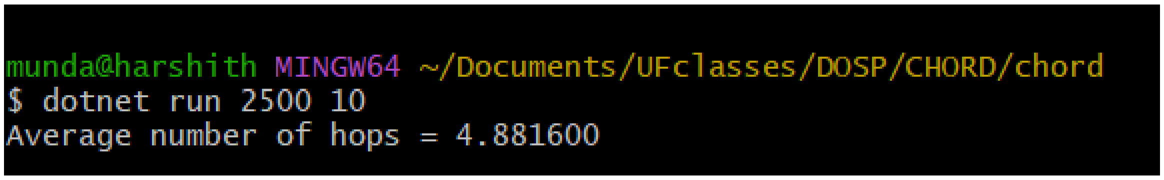
What is working.

- Created chord network ring using create() and dynamically added nodes to the network using join function.
- Finger tables for each node is created.
- The scalable key lookup function is implemented as described in the Chord paper.
- Created a function to lookup keys efficiently.
- Calculated the average hop count using the given formula.

Screenshots of the output



```
munda@harshith MINGW64 ~/Documents/UFclasses/DOSP/CHORD/chord
$ dotnet run 500 10
Average number of hops = 4.858800
```



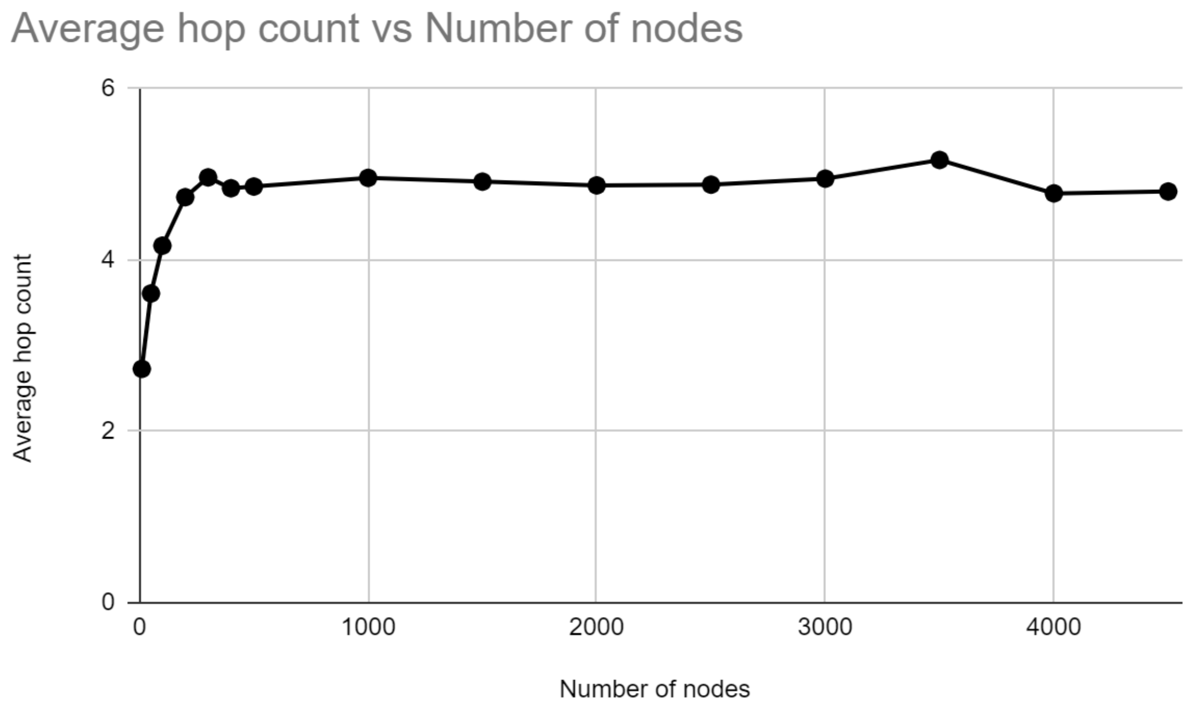
```
munda@harshith MINGW64 ~/Documents/UFclasses/DOSP/CHORD/chord
$ dotnet run 2500 10
Average number of hops = 4.881600
```

Table for average hop count.

We have taken number of requests = 10.

Number of nodes	Average hop count
10	2.73
50	3.61
100	4.168
200	4.734
300	4.965
400	4.838
500	4.858
1000	4.959
1500	4.915
2000	4.8716
2500	4.88
3000	4.95
3500	5.168
4000	4.7784
4500	4.8

Graph for “Number of nodes” vs “Average hop count”



The graph visually represents how the average number of hops in a Chord network changes as the number of nodes in the network increases. It shows the relationship between the size of the Chord network and the efficiency of key lookup operations in terms of hops required.

Largest Network

The largest network we managed to deal with is 4500 nodes. The average hop count calculated was 4.8 when the number of requests passed was 10.