COP5615- Distributed Operating System Principles Fall 2023

Programming Assignment #2

REPORT

PA2_Team14

NAME	UFID	EMAIL
Nitish Chandra Mahesh	**36139637**	n.chandramahesh@ufl.edu
Anuj Papriwal	**37008807**	papriwalanuj@ufl.edu
Pawan Kumar Jagadapuram	**73643747**	pjagadapuram@ufl.edu
Gopi Amarnath Reddy Bekkem	**72188579**	bekkem.g@ufl.edu

Overview:

In this project, we implemented Distributed Hash Table (DHT) systems for peer-to-peer networks using Chord protocol. Chord protocol is implemented using Akka actors, where each actor represents a node in the Chord ring and each node is responsible for a range of keys. We have implemented features like node initialization, finding successors, stabilizing the ring and simulating key lookups to evaluate the performance of the network.

How to Run:

Run the Project using the below command:

Command: `dotnet run <numNodes> <numRequests>`

(where `<numNodes>` is the number of nodes used to create the Chord peer-to-peer system and `<numRequests>` is the number of requests each node makes)

Usage: `dotnet run 5 10`

What is working:

The project employs the Actor model to simulate the Chord protocol. There are two types of actors.

1. **Admin actor**: The Admin actor serves as an entity responsible for simulating the chord network and maintaining experimental statistics. It keeps track of the number of hops taken during key lookup operations by peer actors and generates results with the average hop count.
2. **Node actor**: Each node actor represents a peer in the chord network. Each actor is responsible for maintaining chord responsibilities such as node initialization, lookup operations, stabilization, finger table updates, predecessor and successor handling etc.

Node Initialization:

- Nodes are initialized and added to the Chord ring one by one with each node represented by a ‘Node’ actor.
- The first node plays a key role in initializing the chord ring and sets its predecessor and successor as itself to begin the ring formation.

Lookup Operations:

- The ‘Node’ actors make lookup requests, which are sent as “FindSuccessorNode” messages to further nodes along the ring. These requests eventually find the successor node responsible for a specific key.
- The Chord routing algorithm is used to forward lookup requests to the appropriate nodes.

Stabilization:

- Periodically (every 50 ms), nodes send “Stabilize” to verify their immediate successor nodes.
- They send notifications to their successor nodes to inform them about themselves.

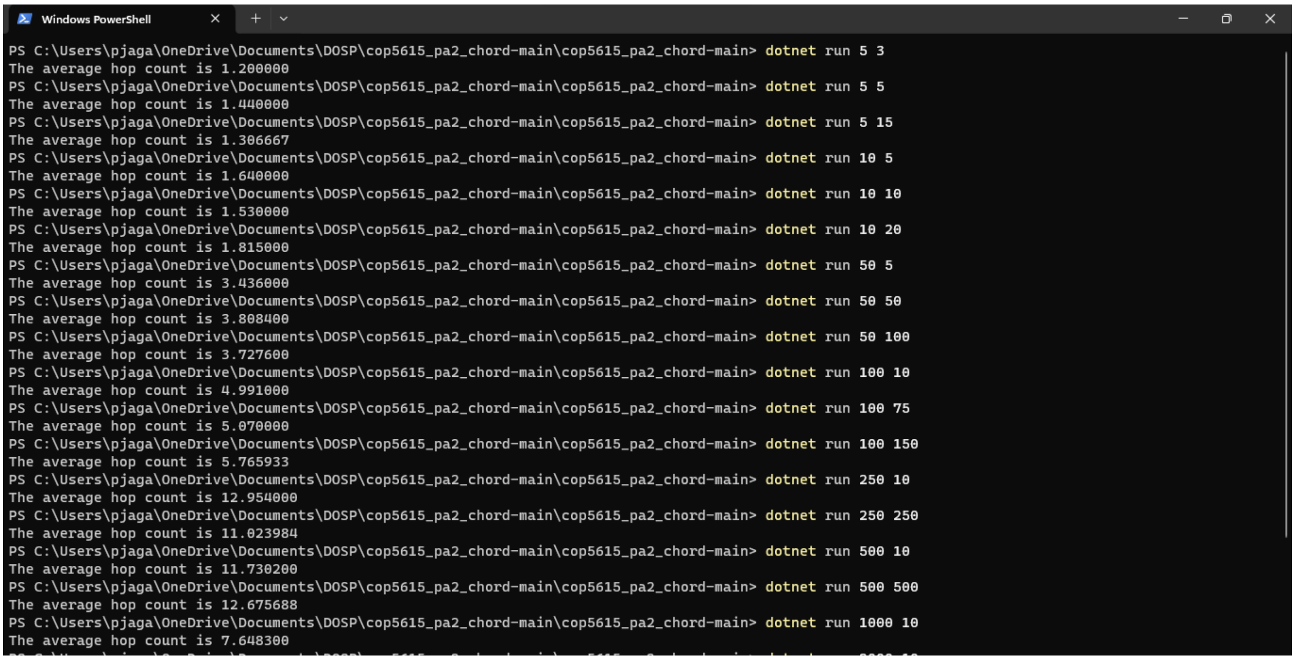
Finger Table Updates:

- “FixFingerTables” messages are sent periodically (every 50 ms) for each node so that the finger tables are updated to include new nodes that may have joined the ring.

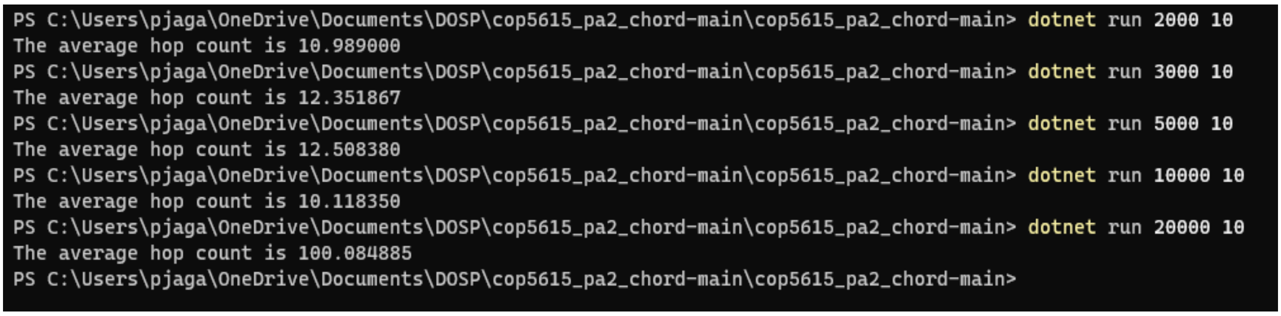
Reporting:

- The ‘Admin’ actor tracks the number of hops taken during lookup operations and calculates the average hop count and outputs it to the console.

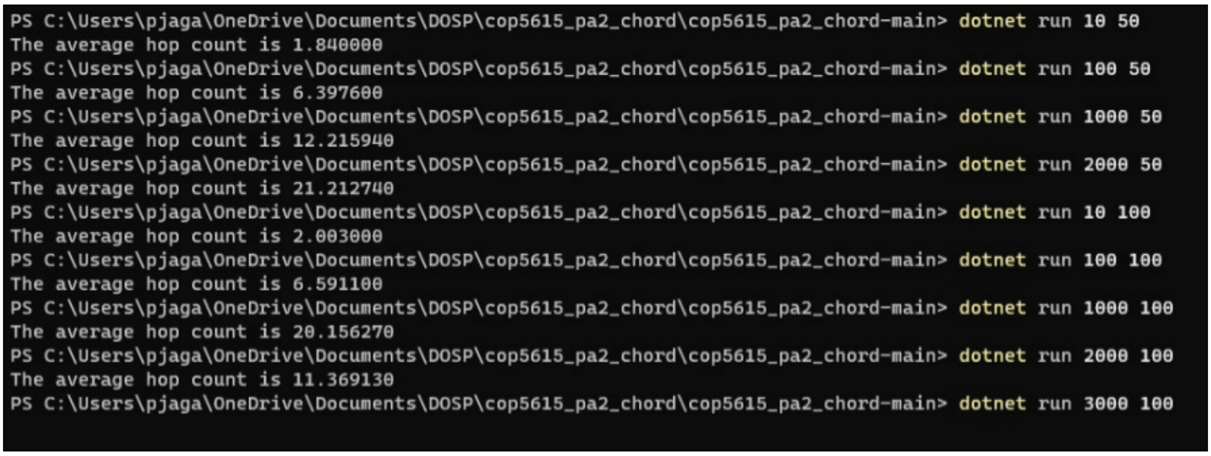
Testing Screenshots:



```
PS C:\Users\pjaga\OneDrive\Documents\DOSP\cop5615_pa2_chord-main\cop5615_pa2_chord-main> dotnet run 5 3
The average hop count is 1.200000
PS C:\Users\pjaga\OneDrive\Documents\DOSP\cop5615_pa2_chord-main\cop5615_pa2_chord-main> dotnet run 5 5
The average hop count is 1.440000
PS C:\Users\pjaga\OneDrive\Documents\DOSP\cop5615_pa2_chord-main\cop5615_pa2_chord-main> dotnet run 5 15
The average hop count is 1.306667
PS C:\Users\pjaga\OneDrive\Documents\DOSP\cop5615_pa2_chord-main\cop5615_pa2_chord-main> dotnet run 10 5
The average hop count is 1.640000
PS C:\Users\pjaga\OneDrive\Documents\DOSP\cop5615_pa2_chord-main\cop5615_pa2_chord-main> dotnet run 10 10
The average hop count is 1.530000
PS C:\Users\pjaga\OneDrive\Documents\DOSP\cop5615_pa2_chord-main\cop5615_pa2_chord-main> dotnet run 10 20
The average hop count is 1.815000
PS C:\Users\pjaga\OneDrive\Documents\DOSP\cop5615_pa2_chord-main\cop5615_pa2_chord-main> dotnet run 50 5
The average hop count is 3.436000
PS C:\Users\pjaga\OneDrive\Documents\DOSP\cop5615_pa2_chord-main\cop5615_pa2_chord-main> dotnet run 50 50
The average hop count is 3.808400
PS C:\Users\pjaga\OneDrive\Documents\DOSP\cop5615_pa2_chord-main\cop5615_pa2_chord-main> dotnet run 50 100
The average hop count is 3.727600
PS C:\Users\pjaga\OneDrive\Documents\DOSP\cop5615_pa2_chord-main\cop5615_pa2_chord-main> dotnet run 100 10
The average hop count is 4.991000
PS C:\Users\pjaga\OneDrive\Documents\DOSP\cop5615_pa2_chord-main\cop5615_pa2_chord-main> dotnet run 100 75
The average hop count is 5.070000
PS C:\Users\pjaga\OneDrive\Documents\DOSP\cop5615_pa2_chord-main\cop5615_pa2_chord-main> dotnet run 100 150
The average hop count is 5.765933
PS C:\Users\pjaga\OneDrive\Documents\DOSP\cop5615_pa2_chord-main\cop5615_pa2_chord-main> dotnet run 250 10
The average hop count is 12.954000
PS C:\Users\pjaga\OneDrive\Documents\DOSP\cop5615_pa2_chord-main\cop5615_pa2_chord-main> dotnet run 250 250
The average hop count is 11.023984
PS C:\Users\pjaga\OneDrive\Documents\DOSP\cop5615_pa2_chord-main\cop5615_pa2_chord-main> dotnet run 500 10
The average hop count is 11.730200
PS C:\Users\pjaga\OneDrive\Documents\DOSP\cop5615_pa2_chord-main\cop5615_pa2_chord-main> dotnet run 500 500
The average hop count is 12.675688
PS C:\Users\pjaga\OneDrive\Documents\DOSP\cop5615_pa2_chord-main\cop5615_pa2_chord-main> dotnet run 1000 10
The average hop count is 7.648300
```



```
PS C:\Users\pjaga\OneDrive\Documents\DOSP\cop5615_pa2_chord-main\cop5615_pa2_chord-main> dotnet run 2000 10
The average hop count is 10.989000
PS C:\Users\pjaga\OneDrive\Documents\DOSP\cop5615_pa2_chord-main\cop5615_pa2_chord-main> dotnet run 3000 10
The average hop count is 12.351867
PS C:\Users\pjaga\OneDrive\Documents\DOSP\cop5615_pa2_chord-main\cop5615_pa2_chord-main> dotnet run 5000 10
The average hop count is 12.508380
PS C:\Users\pjaga\OneDrive\Documents\DOSP\cop5615_pa2_chord-main\cop5615_pa2_chord-main> dotnet run 10000 10
The average hop count is 10.118350
PS C:\Users\pjaga\OneDrive\Documents\DOSP\cop5615_pa2_chord-main\cop5615_pa2_chord-main> dotnet run 20000 10
The average hop count is 100.084885
PS C:\Users\pjaga\OneDrive\Documents\DOSP\cop5615_pa2_chord-main\cop5615_pa2_chord-main>
```



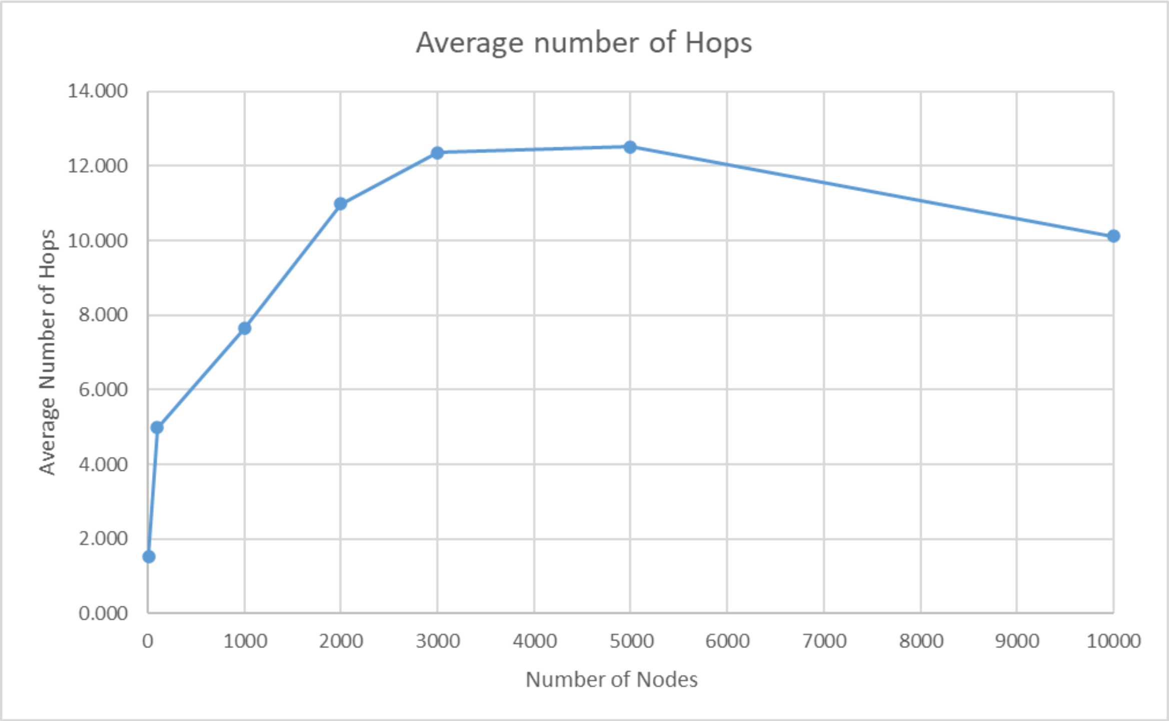
```
PS C:\Users\pjaga\OneDrive\Documents\DOSP\cop5615_pa2_chord\cop5615_pa2_chord-main> dotnet run 10 50
The average hop count is 1.840000
PS C:\Users\pjaga\OneDrive\Documents\DOSP\cop5615_pa2_chord\cop5615_pa2_chord-main> dotnet run 100 50
The average hop count is 6.397600
PS C:\Users\pjaga\OneDrive\Documents\DOSP\cop5615_pa2_chord\cop5615_pa2_chord-main> dotnet run 1000 50
The average hop count is 12.215940
PS C:\Users\pjaga\OneDrive\Documents\DOSP\cop5615_pa2_chord\cop5615_pa2_chord-main> dotnet run 2000 50
The average hop count is 21.212740
PS C:\Users\pjaga\OneDrive\Documents\DOSP\cop5615_pa2_chord\cop5615_pa2_chord-main> dotnet run 10 100
The average hop count is 2.003000
PS C:\Users\pjaga\OneDrive\Documents\DOSP\cop5615_pa2_chord\cop5615_pa2_chord-main> dotnet run 100 100
The average hop count is 6.591100
PS C:\Users\pjaga\OneDrive\Documents\DOSP\cop5615_pa2_chord\cop5615_pa2_chord-main> dotnet run 1000 100
The average hop count is 20.156270
PS C:\Users\pjaga\OneDrive\Documents\DOSP\cop5615_pa2_chord\cop5615_pa2_chord-main> dotnet run 2000 100
The average hop count is 11.369130
PS C:\Users\pjaga\OneDrive\Documents\DOSP\cop5615_pa2_chord\cop5615_pa2_chord-main> dotnet run 3000 100
```

Average Hop Count Results for different test cases:

Number of Nodes	**Number of Requests**	**Average number of Hops**
5	3	1.200
5	5	1.440
5	15	1.306
10	5	1.640
10	10	1.530
10	20	1.815
50	5	3.436
50	50	3.808
50	100	3.727
100	10	4.991
100	75	5.070
100	150	5.765
250	10	12.954
250	250	11.023
500	10	11.730
500	500	12.675
1000	10	7.648
2000	10	10.989

3000	10	12.351
5000	10	12.508
10000	10	10.118
20000	10	100.084
10	50	1.840
100	50	6.397
1000	50	12.215
2000	50	21.212

Graph: Number of Nodes v/s Average Hop Count (Number of Requests =10)



What is the largest network you managed to deal with?

The largest network that we could manage to build and test is for 20000 nodes with each peer making 10 requests. The average hop count we obtained is 100.084. We tried increasing the nodes even further but the Chord ring was taking too long to stabilize.

Assumptions:

After adding all the nodes one by one to the ring, we are waiting for up to 20 seconds to allow all nodes to stabilize and fix their finger tables.