

COP 5615: Fall 2023

Programming Assignment #2

Chord: P2P System and Simulation

TEAM 12

TEAM MEMBERS

Name	UFID	Email
Krishna Priya Karnasula	62991582	k.karnasula@ufl.edu
Sai Krishna Movva	90756144	saikrishnamovva@ufl.edu
Siddharth Chinamuthevi	37301385	Chinamuthevi.s@ufl.edu
Lasya Sree Devabhaktuni	15186666	lasyasre.devabha@ufl.edu

CONTENTS

Serial Number	Content
1.	Introduction
2.	Running the program
3.	Working
4.	Output Screenshots
5.	Table: Average hop counts & various number of nodes
6.	Graph: “Number of nodes” vs “ Average number of hops”
7.	Assumptions
8.	Largest network managed dealt with
9.	Result
10.	Conclusion
11.	Contributions

INTRODUCTION

The main aim of the programming assignment 2 is to implement the Chord protocol using F#. This project consists of two files, a fs file and a fsx file. This project has to work in such a way that when we run the .fs file, it has to call the .fsx file and give the output. We take the inputs as the number of nodes and number of requests and the output will be the created nodes, number of hops, Total number of requests, Average number of hops.

HOW TO RUN YOUR PROGRAM

STEP1: Create a .fsx file which should contain the functions to create a Node, find the Successor node, Join, fetch the Successor node, stabilise, find the fix finger, fetch the Finger successor, update the finger table, update the Successor and to find the key.

STEP2: Once the code has been written then just to run the fsx file we need to give the input in terminal as **dotnet fsi filename.fsx numNodes NumRequests**

Example: dotnet fsi proj2.fsx 5 4

STEP3: In the fs write the code which stores the value of the variables which are mutable and declare the values of the numNodes and numRequest and print the output.

STEP4: To run the fs file the command has to be **dotnet run numberOfNodes NumberOfRequest**

Now, the fsx file will be called from this and need not be executed separately.

Example: dotnet run 5 4

STEP5: In the terminal we get to see the number of nodes created based on the number of requests given, number of hops, Average number of hops.

The formula to calculate the

Average number of hops = $\text{Sum of number of no.of hops for all requests for all nodes} / (\text{numRequests} * \text{numNodes})$

STEP6: We need to create a graph for the number of nodes VS Average number of hops.

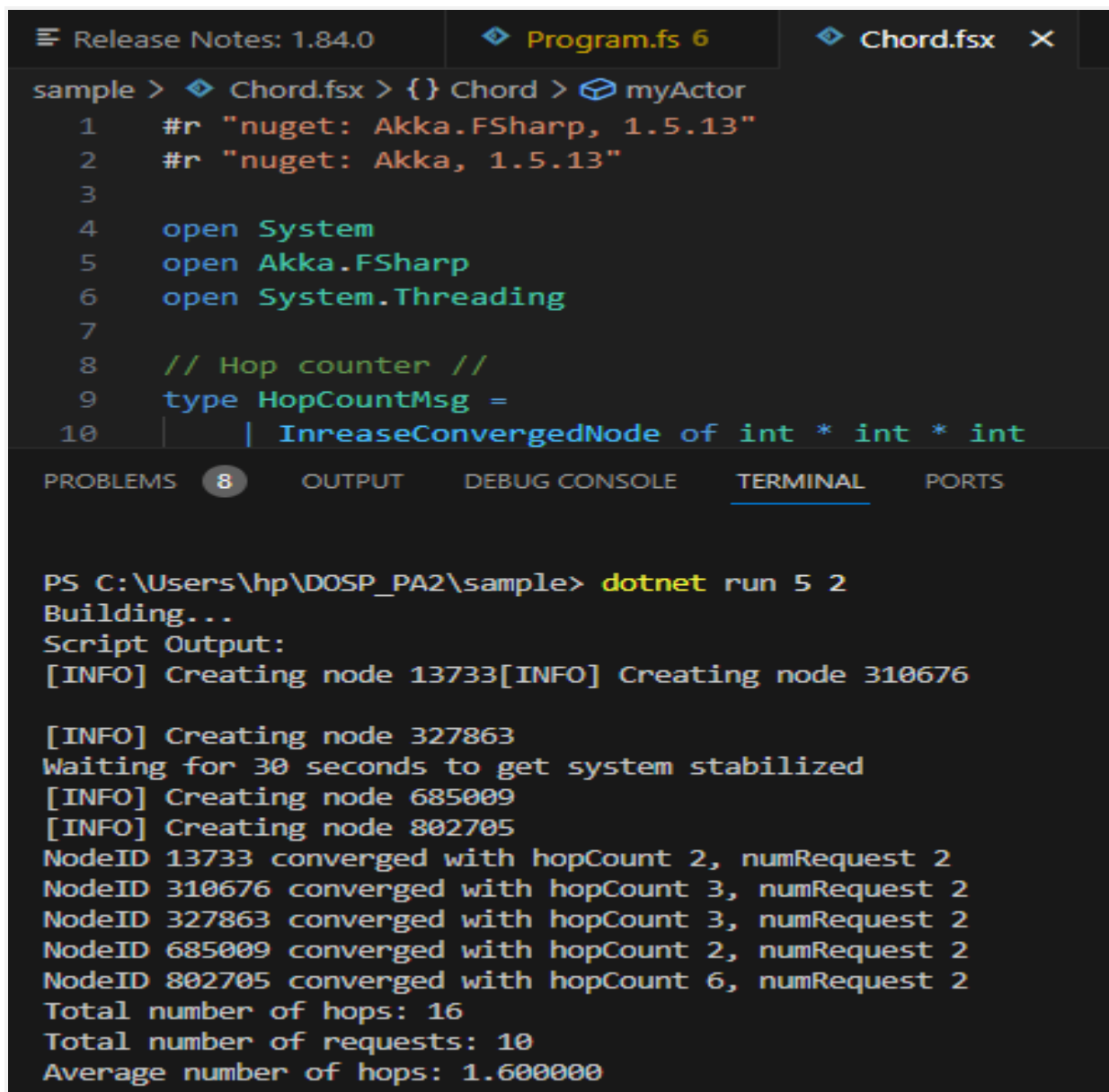
WHAT IS WORKING

We are able to fetch the data efficiently from the Chord Data.

The code effectively simulates a Chord distributed hash table (DHT) using Akka.NET in F#. It achieves the following functionalities:

1. It successfully creates and manages a Chord ring of nodes, allowing them to join the network, stabilise their connections, and perform distributed key lookups.
2. It updates the finger tables when a node joins or leaves and stabilises the system.
3. The hopCounter actor accurately tracks the number of hops required for convergence during key lookup operations and calculates the average hop count. This feature provides valuable insights into the efficiency of the Chord DHT simulation.
4. In summary, the code delivers a functional Chord DHT simulation that enables the creation of a network, node operations, and hop count analysis, making it a valuable tool for understanding and analysing Chord-based distributed systems.

SCREENSHOTS OF THE OUTPUT



The screenshot displays the Visual Studio Code interface. The top bar shows the 'Release Notes: 1.84.0' and two open files: 'Program.fs 6' and 'Chord.fsx'. The 'Chord.fsx' file is active, showing F# code for a distributed system simulation. The code includes namespace imports for System, Akka.FSharp, and System.Threading, a comment for a hop counter, and a type definition for HopCountMsg. The output window at the bottom shows the command 'dotnet run 5 2' being executed, followed by build and script output messages. The output details the creation of five nodes, their convergence with hop counts and request numbers, and the final statistics: 16 total hops, 10 total requests, and an average of 1.6 hops per request.

```
sample > Chord.fsx > {} Chord > myActor
1  #r "nuget: Akka.FSharp, 1.5.13"
2  #r "nuget: Akka, 1.5.13"
3
4  open System
5  open Akka.FSharp
6  open System.Threading
7
8  // Hop counter //
9  type HopCountMsg =
10 | IncreaseConvergedNode of int * int * int
```

PROBLEMS 8 OUTPUT DEBUG CONSOLE TERMINAL PORTS

```
PS C:\Users\hp\DOSP_PA2\sample> dotnet run 5 2
Building...
Script Output:
[INFO] Creating node 13733[INFO] Creating node 310676

[INFO] Creating node 327863
Waiting for 30 seconds to get system stabilized
[INFO] Creating node 685009
[INFO] Creating node 802705
NodeID 13733 converged with hopCount 2, numRequest 2
NodeID 310676 converged with hopCount 3, numRequest 2
NodeID 327863 converged with hopCount 3, numRequest 2
NodeID 685009 converged with hopCount 2, numRequest 2
NodeID 802705 converged with hopCount 6, numRequest 2
Total number of hops: 16
Total number of requests: 10
Average number of hops: 1.600000
```

Fig 1: numNodes = 5 and numRequests = 2

The image shows a Visual Studio Code editor window with two tabs: 'Program.fs 6' and 'Chord.fsx'. The 'Chord.fsx' tab is active, displaying F# code for a Chord simulation. The code defines a `main` function that initializes `nodeIDs` and `sortedNodeIDs`, spawns nodes, and runs a loop to simulate requests. The terminal output shows the results of the simulation for 50 nodes and 2 requests.

```
sample > Chord.fsx > {} Chord > main
242 let nodeIDs : int array = Array.init count = numNodes (fun _
243
244 let sortedNodeIDs : int array = Array.sort nodeIDs
245 // Spawn nodes
246 let nodeIDs : int array = Array.create count = numNodes -1;
247 let nodeRefs : Akka.Actor.IActorRef array = Array.create count = numNodes -1;
248 let mutable i : int = 0;
249 while(i < numNodes) do
250     try
251         let nodeID : int = sortedNodeIDs.[i]
252
253         nodeRefs.[i] <- spawn actorFactory = system name = (s
254         if(i = 0) then
255             nodeRefs.[i] <| Create
256         else
```

PROBLEMS 8 OUTPUT DEBUG CONSOLE TERMINAL PORTS

```
NodeID 822198 converged with hopCount 5, numRequest 2
NodeID 831862 converged with hopCount 9, numRequest 2
NodeID 842621 converged with hopCount 4, numRequest 2
NodeID 875522 converged with hopCount 8, numRequest 2
NodeID 895250 converged with hopCount 7, numRequest 2
NodeID 944926 converged with hopCount 7, numRequest 2
NodeID 955790 converged with hopCount 5, numRequest 2
NodeID 959808 converged with hopCount 6, numRequest 2
NodeID 971152 converged with hopCount 8, numRequest 2
NodeID 975222 converged with hopCount 11, numRequest 2
NodeID 1014953 converged with hopCount 7, numRequest 2
Total number of hops: 280
Total number of requests: 100
Average number of hops: 2.800000

PS C:\Users\hp\DOSP_PA2\sample>
```

Fig 2: numNodes = 50, numRequests = 2

```
Release Notes: 1.84.0 | Program.fs 6 | Chord.fsx x
sample > Chord.fsx > {} Chord > main
242 let nodeIDs : int array = Array.init count = numNodes
243
244 let sortedNodeIDs : int array = Array.sort nodeIDs
245 // Spawn nodes
246 let nodeIDs : int array = Array.create count = numNodes
247 let nodeRefs : Akka.Actor.IActorRef array = Array.create
248 let mutable i : int = 0;
249 while(i < numNodes) do
250     try
251         let nodeId : int = sortedNodeIDs.[i]
252
253         nodeRefs.[i] <- spawn actorFactory = system na
254         if(i = 0) then
255             nodeRefs.[i] <! Create
256         else

```

PROBLEMS 8 OUTPUT DEBUG CONSOLE TERMINAL PORTS

```
NodeID 956773 converged with hopCount 10, numRequest 2
NodeID 966074 converged with hopCount 8, numRequest 2
NodeID 971944 converged with hopCount 2, numRequest 2
NodeID 999774 converged with hopCount 10, numRequest 2
NodeID 1009693 converged with hopCount 9, numRequest 2
NodeID 1010419 converged with hopCount 12, numRequest 2
NodeID 1016088 converged with hopCount 8, numRequest 2
NodeID 1020878 converged with hopCount 11, numRequest 2
NodeID 1033710 converged with hopCount 8, numRequest 2
NodeID 1037119 converged with hopCount 11, numRequest 2
NodeID 1045276 converged with hopCount 12, numRequest 2
Total number of hops: 533
Total number of requests: 200
Average number of hops: 2.665000

PS C:\Users\hp\DOSP_PA2\sample>
```

Fig 3:numNodes = 100, numRequests = 2

```
Release Notes: 1.84.0 | Program.fs 6 | Chord.fsx X
sample > Chord.fsx > {} Chord > main
242 let nodeIDs : int array = Array.init count = numNodes (f
243
244 let sortedNodeIDs : int array = Array.sort nodeIDs
245 // Spawn nodes
246 let nodeIDs : int array = Array.create count = numNodes
247 let nodeRefs : Akka.Actor.IActorRef array = Array.create
248 let mutable i : int = 0;
249 while(i < numNodes) do
250     try
251         let nodeID : int = sortedNodeIDs.[i]
252
253         nodeRefs.[i] <- spawn actorFactory = system name
254         if(i = 0) then
255             nodeRefs.[i] <! Create
256         else

```

PROBLEMS 8 OUTPUT DEBUG CONSOLE TERMINAL PORTS

```
NodeID 1036845 converged with hopCount 9, numRequest 2
NodeID 1037532 converged with hopCount 9, numRequest 2
NodeID 1039353 converged with hopCount 15, numRequest 2
NodeID 1041203 converged with hopCount 15, numRequest 2
NodeID 1041537 converged with hopCount 8, numRequest 2
NodeID 1043427 converged with hopCount 11, numRequest 2
NodeID 1043876 converged with hopCount 10, numRequest 2
NodeID 1046028 converged with hopCount 11, numRequest 2
NodeID 1046123 converged with hopCount 10, numRequest 2
NodeID 1046916 converged with hopCount 13, numRequest 2
NodeID 1048533 converged with hopCount 5, numRequest 2
Total number of hops: 6273
Total number of requests: 2000
Average number of hops: 3.136500

PS C:\Users\hp\DOSP_PA2\sample>
```

Fig 4: numNodes = 1000, numRequests = 2

TABLE: NUMBER OF NODES VS THE AVERAGE HOP COUNT

Number of Nodes	Number of Requests	Average Number of Hops
5	2	1.3
10	2	1.55
20	2	2.275
30	2	2.766667
40	2	2.71
50	2	2.74
70	2	2.75
100	2	3.06
5	9	1.177778
10	9	1.677778
20	9	2.166667
50	9	2.846667
200	9	3.179444
1000	9	3.386222

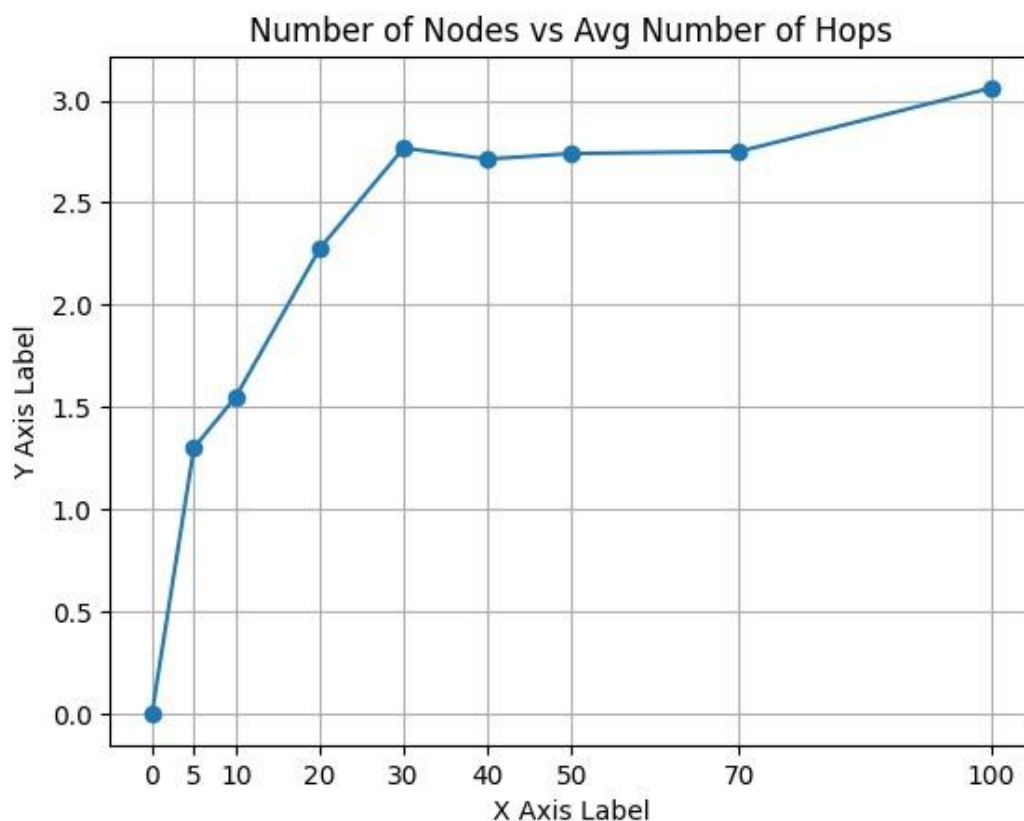
Number of Requests that we have considered are 2 and 9

GRAPH: NUMBER OF NODES VS AVERAGE HOP COUNT

For the Graph to be plotted for Number of nodes VS Average hop count we need to have a constant Request for better comparison. The maximum number of nodes that we have considered are 1000 and the graph plotted looks similar to a logarithmic function(i.e., n vs $\log n$).

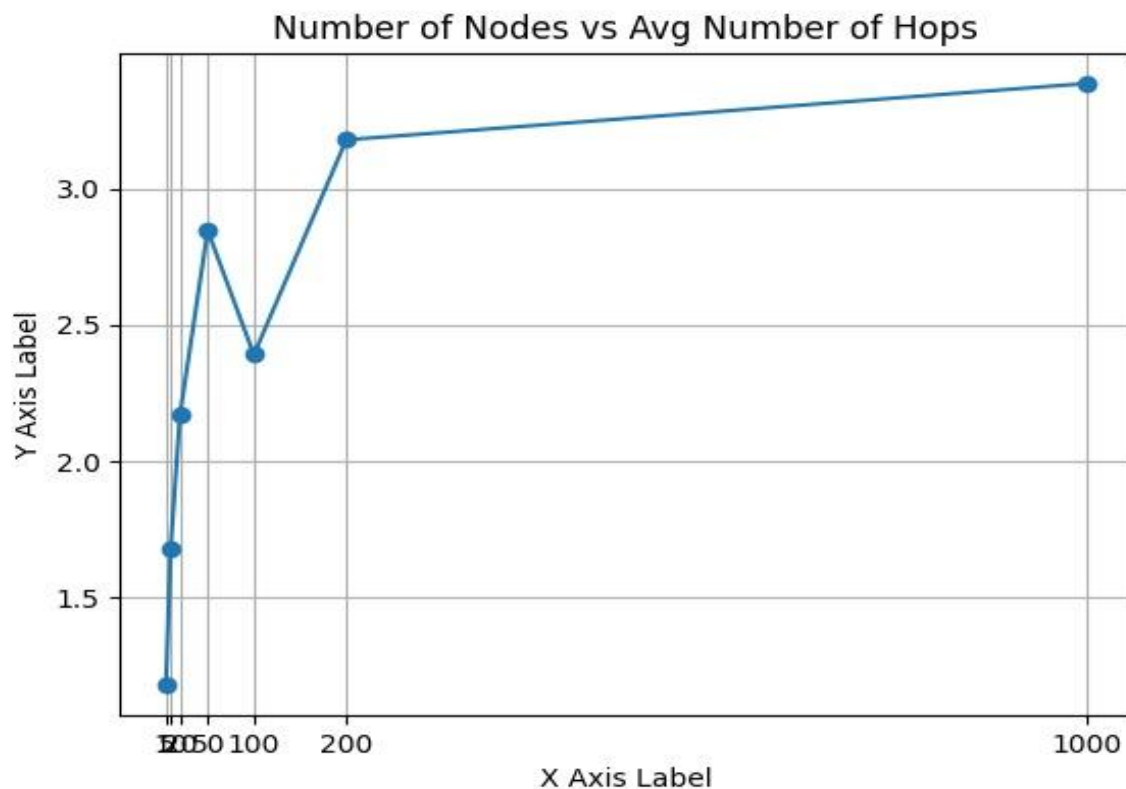
Graph1: numRequests = 2:

X axis→Number of Nodes, Y axis →Average Number of Hops



Graph2: numRequests = 9:

X axis→Number of Nodes, Y axis →Average Number of Hops



ASSUMPTIONS

For plotting the graph

- 1) We assumed the number of requests(numRequest) as 2 for graph1.
- 2) We assumed the number of requests(numRequest) as 9 for graph2.

For the purpose of handling distributed data, the Chord P2P simulation code assumes a decentralised distributed hash table (DHT) system. It is intended to emulate several activities such as node joining, key-based lookups, and query processing, and to

represent nodes using m-bit identifiers. In order to manage concurrent messaging between nodes and effectively determine the average number of hops needed for routing in the Chord network, the code makes use of the actor model. Although the simulation offers insightful information about how Chord-based systems behave, it makes node building easier by providing arbitrary identifiers, which might not accurately reflect real-world situations. Nonetheless, it is an essential tool for learning about and doing experiments with chord-based DHT networks.

LARGEST NETWORK

We have considered networks of various sizes starting from 5 until 1000. Hence, the largest network managed that we are dealing with in this project is 1000 nodes at once with the number of requests as 2.

RESULT

The Chord P2P simulation project evaluates node behaviour, key lookups, and network stability to shed light on Chord-based peer-to-peer networks. Metrics like average hop count are measured, which helps with efficiency and scalability assessments. It supports learning and the development of applications by being used for testing and educational purposes. We have obtained a logarithmic graph as a result of plotting the number of nodes vs average number of hops. In conclusion, it improves knowledge of Chord networks for both theoretical and applied study.

CONCLUSION

The project's conclusion would highlight the lessons acquired by simulating Chord-based peer-to-peer networks. It would emphasise the importance of indicators such as average hop count in evaluating network efficiency and scalability. The conclusion might go over how this simulation tool promotes learning and development, as well as its potential for real-world applications. In short, it would highlight the project's contributions to Chord network research and practical use.

CONTRIBUTIONS

Krishna Priya Karnasula:

- Played a key role in code development.
- Her responsibilities included developing node and hop counter actors, managing messages, and simulating network processes such as node joins, finger table maintenance, and key lookups.
- Also worked on the project's report, which summarised the simulation tool's findings, limits, and prospective use cases.

Sai Krishna Movva:

- Focused on simulation setup and experimentation.
- Contributions improved the project's overall impact and aided in the analysis of the data to produce insightful findings.
- His approach entailed conducting simulations, collecting data, and analysing the findings, specifically calculating the average hop count for various scenarios.

Siddharth Chinamuthevi:

- In charge of research and documentation.
- Helped with parameter configuration, such as defining network capacity, request number, and identifier space.
- Created graphs and plots using Python scripts based on simulation data, particularly in the graph-plotting challenge that showed the correlation between the average number of hops and the number of nodes. provided insightful data regarding network performance.

Lasya Sree Devabhaktuni:

- In charge of writing the F# code for the Chord network simulation utilising the actor model, creating actors to represent Chord nodes, implementing key Chord network functionalities such as node joins, stabilisation, and distributed key lookups.
- Involved in the visualisation of the results.
- In charge of documenting assumptions, key ideas, and explaining how the simulation code corresponds to Chord network theory.