

Programming Assignment 2

PA2 Team1

November 2, 2023

1 Team Information

Name	UFID	email id
Harshit Lohaana	7615-8695	h.lohaana@ufl.edu
Vashist Hegde	8721-8376	vashisthegde@ufl.edu
Rahul Mora	4577-9236	mora.rahul@ufl.edu
Sushmitha Kasireddy	5336-8700	s.kasireddy@ufl.edu

2 How to Run the Code

After having the .net SDK, we can use the following commands to run the programs:

- To install Akka in the current working directory of the project
`dotnet new install "Akka.templates:.*"`
- To run the project:
`dotnet run {number of nodes} {number of messages}`

3 Description of Code Structure in the file Program.fs (What is working)

Our code has 2 major actors: Node and Simulator.

3.1 Node Actor

Node is to represent each node in the Chord ring, and has the following properties:

- iD
- currentFinger
- FingerTable
- predecessorId
- successorId
- hopCount

- mssgCount

Node actor implements the following methods:

- requestData
- request
- requestInfo
- resultInfo
- fixFinger
- fixFingerRes
- stabilize
- stabilizeRes
- initialization

3.2 Simulator Actor

The Simulator actor is to implement the Chord ring simulation. It maintains the following properties:

- hopCount
- completedNodes
- nodeIdHopsDirectory

Simulator actor implements the following methods:

- Create
- Join
- CreateChord

4 Team Contributions

- **Sushmitha Kasireddy** - Responsible for testing end-to-end functionality and helped with the report.

- **Rahul Mora** - Develop of algorithm for joining nodes, creating nodes and update fingertables on joining if needed.
- **Vashist Hegde** - Responsible for experimenting with the code, including the data collection, plotting, and writing the report.
- **Harshit Loha** - Creation of actor simulator and node simulator for simulation processes. Export and collection of data from the code.

5 Execution Result Screenshots

5.1 Abnormal Results

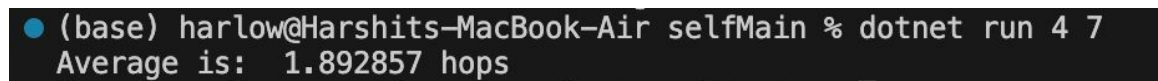
We did not notice any abnormal results during the execution of our code.

5.2 Bugs, missing items, limitations of the code

Bugs: Graceful termination is not handled in case the input exceeds the below mentioned capacity.

Limitation: The maximum number of nodes our program can accomodate is 1000.

5.3 Instances of the code working

A terminal window with a dark background. The prompt is a blue circle followed by "(base) harlow@Harshits-MacBook-Air selfMain %". The command "dotnet run 4 7" has been entered. The output is "Average is: 1.892857 hops".

```
● (base) harlow@Harshits-MacBook-Air selfMain % dotnet run 4 7
Average is: 1.892857 hops
```

Figure 1: Output for 4 nodes and 7 requests

```

● (base) harlow@Harshits-MacBook-Air selfMain % dotnet run 5 3
Average is: 1.466667 hops
● (base) harlow@Harshits-MacBook-Air selfMain % dotnet run 10 6
Average is: 2.133333 hops
● (base) harlow@Harshits-MacBook-Air selfMain % dotnet run 6 7
Average is: 1.976190 hops
● (base) harlow@Harshits-MacBook-Air selfMain % dotnet run 6 7
Average is: 1.833333 hops
● (base) harlow@Harshits-MacBook-Air selfMain % dotnet run 7 9
Average is: 1.857143 hops
● (base) harlow@Harshits-MacBook-Air selfMain % dotnet run 7 9
Average is: 2.285714 hops
● (base) harlow@Harshits-MacBook-Air selfMain % dotnet run 2 10
Average is: 1.500000 hops
● (base) harlow@Harshits-MacBook-Air selfMain % dotnet run 3 10
Average is: 1.666667 hops
● (base) harlow@Harshits-MacBook-Air selfMain % dotnet run 4 10
Average is: 1.750000 hops
● (base) harlow@Harshits-MacBook-Air selfMain % dotnet run 5 10
Average is: 2.400000 hops
● (base) harlow@Harshits-MacBook-Air selfMain % dotnet run 6 10
Average is: 2.216667 hops
● (base) harlow@Harshits-MacBook-Air selfMain % dotnet run 7 10
Average is: 2.071429 hops
● (base) harlow@Harshits-MacBook-Air selfMain % dotnet run 8 10
Average is: 1.875000 hops
● (base) harlow@Harshits-MacBook-Air selfMain % dotnet run 9 10
Average is: 2.555556 hops
● (base) harlow@Harshits-MacBook-Air selfMain % dotnet run 10 10
Average is: 2.950000 hops
● (base) harlow@Harshits-MacBook-Air selfMain % dotnet run 20 10
Average is: 3.655000 hops
● (base) harlow@Harshits-MacBook-Air selfMain % dotnet run 30 10
Average is: 2.703333 hops
● (base) harlow@Harshits-MacBook-Air selfMain % dotnet run 40 10
Average is: 3.170000 hops
● (base) harlow@Harshits-MacBook-Air selfMain % dotnet run 50 10
Average is: 3.416000 hops
● (base) harlow@Harshits-MacBook-Air selfMain % dotnet run 60 10
Average is: 3.371667 hops
● (base) harlow@Harshits-MacBook-Air selfMain % dotnet run 70 10
Average is: 3.568571 hops

```

Figure 2: Output of multiple runs

6 Result Table

Number of nodes	Number of Requests	Average number of Hops
2	10	1.5
3	10	1.667
4	10	1.75
5	10	2.4
6	10	2.216
7	10	2.071
8	10	1.875
9	10	2.556
10	10	2.95
20	10	3.65
30	10	2.703
40	10	3.17
50	10	3.416
60	10	3.371
70	10	3.568
80	10	4.077
90	10	4.065
100	10	3.692
1000	10	5.5884

7 Plot of Number of Nodes vs. Average number of hops

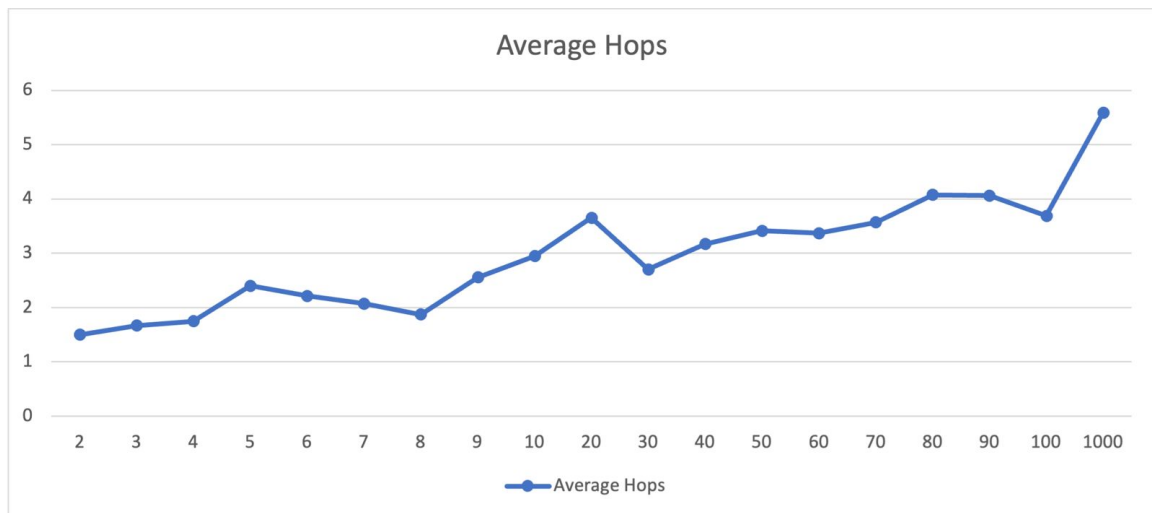


Figure 3: Plot of Number of nodes versus Average number of Hops for 10 requests