Project 3

This is project 3 in the course COP5615 Distributed Operating System Principles.

In this project I implemented the Chord protocol for a peer-to-peer distributed hash table. The project is exclusively written in Pony with the help of actor model functionality.

The simulation models a peer-to-peer network with nodes organized in a ring, where each node can lookup keys, join, and manage routing information. The Chord protocol is known for its scalability, with logarithmic performance on node lookups and hops.

Overview:

The simulation initializes a Chord network with a specified number of nodes, each responsible for locating a portion of the distributed key space. Each node maintains a finger table, which helps in efficiently routing requests to find the node responsible for a given key. Nodes periodically issue requests to locate successors and resolve keys, with the average hop count being reported as a measure of performance.

What is working?

All the requirements stated in the project description has met and my implementation reflects the API calls specified in paper. The program simulates the nodes in the network -- using one actor for each peer -- where you can successfully store keys on each node in the network and perform key lookup.

Features:

Node Join: Nodes can join the network and find their positions based on a hash function.

Finger Table Updates: Each node maintains a finger table, allowing efficient routing and lookup.

Key Lookup: Nodes can lookup keys, find successors, and report hops needed to resolve each request.

Randomized Load Testing: Nodes initiate multiple requests to simulate a real-world DHT load.

Hops Tracking: Tracks and reports the average hops required to resolve requests.

In this simulation, hashing plays a crucial role in distributing keys across nodes in the Chord network. Specifically, a simple multiplicative hash function is used to map each node's unique identifier and keys onto a fixed-size identifier space. This ensures that each node is responsible for a specific segment of this space, allowing for efficient, consistent routing and lookup.

Hashing in Chord:

The Chord protocol typically relies on a consistent hashing approach, meaning that the hash function maps both nodes and data to a circular identifier space (a "ring"). This ensures that:

Load is balanced and Scalability is supported.

This function uses the Knuth's multiplicative method (with the golden ratio) to distribute values across the identifier space. The bits parameter specifies the number of bits in the identifier space, allowing flexibility in network size.

Author:

Anisa Shaik (UFID – 49373585)

Running the program

You can run the program for a single input with the following command:

.\project3 <numNodes> <numrequests>

The largest network managed to deal with

numNodes	numrequests	**Average hops**
3000	**400**	**2.0796**

```
PS C:\Users\shaik\OneDrive\Desktop\PONY\ChordP2P\project3>  .\project3 200 4
Starting Chord simulation with 200 nodes and 4 requests per node
Total requests completed: 800
Total hops: 1583
Average hops per request: 1.97875
PS C:\Users\shaik\OneDrive\Desktop\PONY\ChordP2P\project3>  .\project3 2000 100
Starting Chord simulation with 2000 nodes and 100 requests per node
Total requests completed: 200000
Total hops: 420581
Average hops per request: 2.1029
PS C:\Users\shaik\OneDrive\Desktop\PONY\ChordP2P\project3>  .\project3 3000 200
Starting Chord simulation with 3000 nodes and 200 requests per node
Total requests completed: 600000
Total hops: 1241156
Average hops per request: 2.06859
PS C:\Users\shaik\OneDrive\Desktop\PONY\ChordP2P\project3>  .\project3 3000 400
Starting Chord simulation with 3000 nodes and 400 requests per node
Total requests completed: 1200000
Total hops: 2495592
Average hops per request: 2.07966
```