DOSP PROJECT 3

Team Members

- Manav Mishra
- Shubham Manoj Singh

What is Working

- The Chord DHT implementation allows nodes to join a ring topology.
- Each node maintains a finger table for efficient key lookups.
- Nodes can handle requests to find keys and locate successors/predecessors.
- The HopCounter actor tracks the total hops and requests, calculating average hops when all nodes converge.
- Randomly generated node IDs ensure unique identification within the network.

Largest Network Managed

- The largest network successfully managed consists of **800** nodes.

Running the Code

To execute the Chord DHT implementation, use the following command:

Running the Code

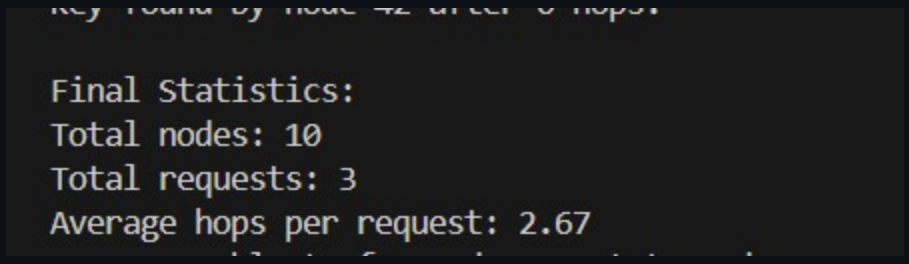
To execute the Chord DHT implementation, use the following command:

```
./dosp3 10 3
```

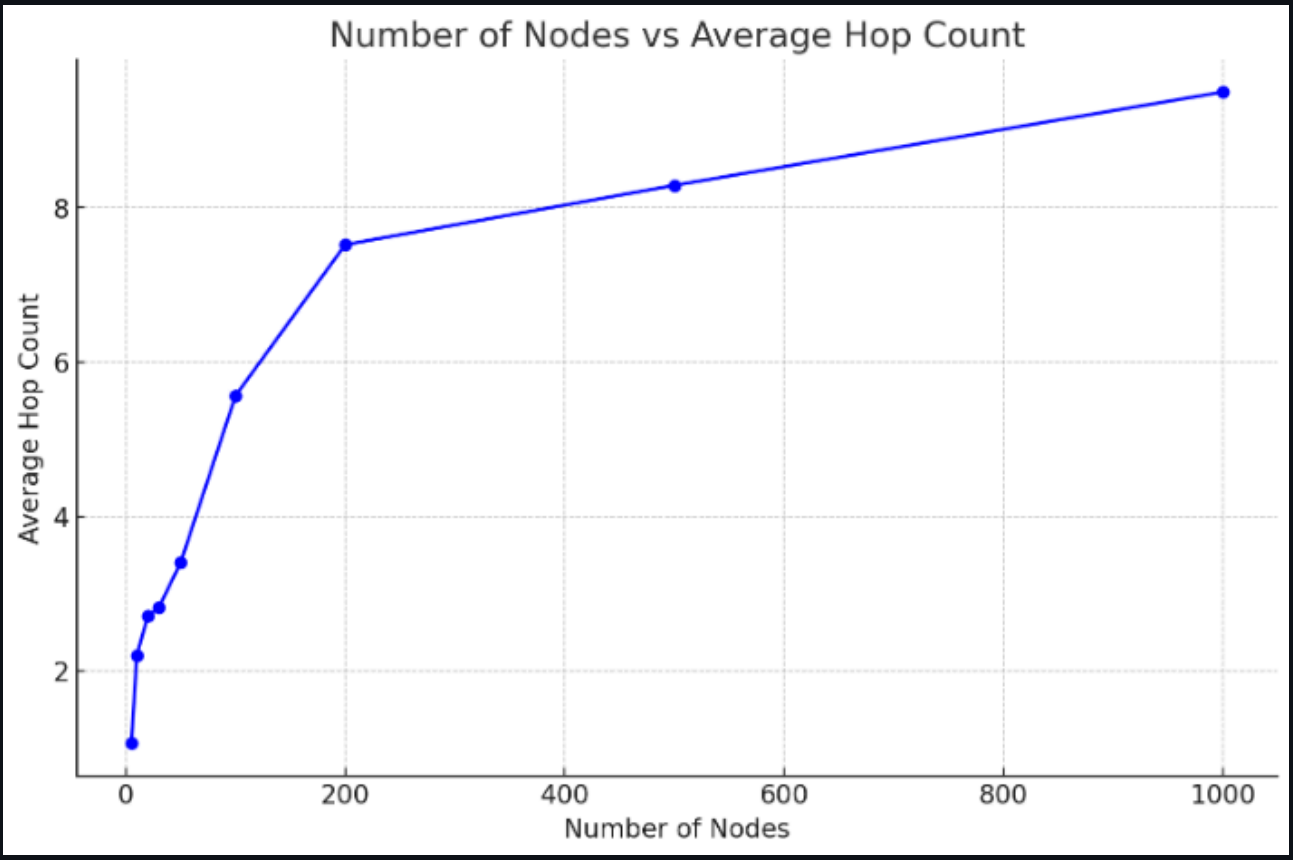
Parameters

- `10` : Number of nodes in the Chord network.
- `3` : Maximum number of requests each node will make.

Make sure to adjust the parameters as needed for different network sizes and request limits.



Number of Nodes VS Average Hop Count



Chord Distributed Hash Table (DHT) Code Explanation

Overview of Chord DHT

Chord is a scalable and efficient distributed hash table (DHT) that allows nodes to efficiently locate keys in a distributed environment. Each node in a Chord network is assigned an identifier (ID), and keys are hashed to these IDs. The network is structured as a circular ring, facilitating efficient lookups.

Key Components

1. Actors

The code uses an actor model for concurrency and message passing, consisting of three main actors: `ChordNode`, `HopCounter`, and `Main`.

2. ChordNode Actor

- **Properties**:
  - `_id` : Unique identifier of the node.
  - `_hop_counter` : Reference to the `HopCounter` actor for tracking metrics.
  - `_predecessor` and `_successor` : References to neighboring nodes.
  - `_finger_table` : An array maintaining pointers to nodes for efficient routing.
  - `_total_hops` , `_requests` : Counters for monitoring performance.
  - `_m` : Number of bits used for ID space.
  - `_total_space` : Total number of possible IDs.
- **Key Methods**:
  - `create()` : Initializes the node, sets up its finger table, and initializes counters.
  - `start_query()` : Begins querying for keys as long as the maximum number of requests hasn't been reached.
  - `query_request()` : Generates a random key and initiates the search for it.
  - `find_next_key()` : Recursively finds the node responsible for a given key, forwarding requests as necessary.
  - `join_ring()` : Allows a new node to join the existing Chord network.
  - `stabilize()` : Periodically checks and maintains the consistency of predecessor/successor relationships.

3. HopCounter Actor

- **Properties**:
  - Tracks total hops and requests across all nodes.
  - Keeps a count of converged nodes.
  - Maintains an array of all `ChordNode` instances.
- **Key Methods**:
  - `node_converged()` : Updates metrics when a node successfully completes its requests.
  - `print_avg_hops()` : Calculates and prints the average number of hops for requests.
  - `add_node()` : Adds a node to the system and retrieves its ID via message passing.

4. Main Actor

- **Functionality**:
  - Initializes the environment, including the number of nodes and requests.
  - Creates nodes with unique IDs and establishes the Chord network.
  - Initiates querying after the network is set up.

How It Works

1. **Initialization**:
   - The `Main` actor creates a specified number of `ChordNode` instances with unique IDs.
   - Each node initializes its finger table and other state variables.
2. **Joining the Ring**:
   - The first node acts as the initial point of the network. Other nodes join by invoking the `join_ring()` method, which locates their position in the ring.
3. **Querying**:
   - Each node starts querying for random keys. The process involves:
     - Generating a random key.
     - Using the finger table to efficiently find the node responsible for that key.
     - Forwarding the request through the network until the key is located or the maximum requests are reached.
4. **Performance Monitoring**:
   - The `HopCounter` actor tracks performance metrics, providing insights into the average number of hops taken to find keys.

Conclusion

This implementation of a Chord DHT provides a robust framework for understanding distributed systems. It emphasizes efficient routing and scalability, essential for managing large networks of nodes. Each component is designed to work concurrently, handling requests and maintaining the network structure dynamically.