INTERNATIONAL INSTITUTE OF INFORMATION TECHNOLOGY

HYDERABAD

Distributed Systems
Project - Chord Replication

November 28, 2024

Project Team ID: 60

Team

Name	RollNumber
Shiva Shankar Gande	2023202005
Sachin Hegde	2023202027

1. Introduction

1.1 Problem Statement

As the volume of data increases in modern computing systems, distributed systems have become essential for providing scalable and reliable solutions. Centralized systems often suffer from issues such as single points of failure, limited scalability, and bottlenecks. Peer-to-peer (P2P) systems, like the Chord Distributed Hash Table (DHT), offer a decentralized alternative for managing distributed storage systems. However, ensuring fault tolerance in such systems remains a significant challenge.

This project focuses on implementing Chord, a decentralized protocol for efficient data lookup, and enhancing it with replication mechanisms to improve fault tolerance and ensure data availability even in the presence of node failures.

1.2 Objectives

- Implement the Chord DHT protocol to support efficient storage and retrieval of key-value pairs.
- Introduce data replication to improve system fault tolerance.
- Allow dynamic node membership (nodes joining and leaving the network) while maintaining consistency and stability.
- Develop a simulation environment to analyze the performance and fault tolerance capabilities of the system.

1.3 Scope

The scope of this project includes:

1. Implementing a Chord ring using consistent hashing.
2. Designing and integrating replication strategies to store multiple copies of data.
3. Handling node failures gracefully without affecting system availability.
4. Simulating the behavior of the system under dynamic conditions (e.g., node churn, failures).
5. Measuring system performance in terms of lookup efficiency, fault tolerance, and scalability.

---

2. Background Theory

2.1 Distributed Hash Tables (DHTs)

DHTs are a class of decentralized storage systems that use a hash function to map data (keys) to specific nodes. Unlike traditional databases, DHTs are designed for environments where nodes frequently join and leave.

Key properties of DHTs:

- **Scalability:** Can handle a large number of nodes.
- **Fault Tolerance:** Can recover from node failures dynamically.
- **Efficiency:** Minimize the number of hops required for lookups.

2.2 Chord Protocol

2.2.1 Definition

Chord is a DHT protocol designed for peer-to-peer systems. It organizes nodes into a logical ring topology, where each node is assigned a unique identifier (ID) in a circular keyspace.

2.2.2 Logical Ring Structure

- Nodes are arranged in a circular identifier space of size 2^M, where $M$is the number of bits in the identifier.
- Each node is responsible for keys in the range $[Predecessor.ID + 1, Node.ID]$.

2.2.3 Key Operations

1. **Lookup:** Locate the node responsible for a given key.
2. **Join:** Add a new node to the network.
3. **Leave:** Remove a node while ensuring data integrity.

2.3 Consistent Hashing

Consistent hashing is used to distribute data among nodes, ensuring minimal disruption when nodes join or leave. Each key is hashed to an identifier, and the key is assigned to the successor node.

2.4 Finger Tables

Each node maintains a routing table called the finger table, which reduces the lookup time to $O(logN)$.

2.5 Replication in Chord

Replication ensures fault tolerance by storing multiple copies of data across several nodes.

---

3. Base Implementation: Chord Without Replication

3.1 Design

The base implementation involves the following:

- **Node:** Represents participants in the Chord ring.
- **Finger Table:** Efficient routing mechanism for lookups.
- **Successor/Predecessor Pointers:** Maintain connectivity of the ring.
- **Lookup Operation:** Ensures efficient location of the responsible node.

3.2 Key Algorithms

1. **Lookup:** Locate the successor node for a given key using the finger table.
2. **Join:** Integrate a new node into the Chord ring by updating finger tables and pointers.

3. **Stabilization:** Periodic checks to ensure the ring structure remains consistent.

3.3 Implementation

The base implementation was developed in Python, using consistent hashing to construct the Chord ring. Each node maintained a finger table and successor/predecessor pointers.

---

4. Extension: Chord with Replication

4.1 Problem Solved

The base implementation lacked fault tolerance. Data unavailability occurred when nodes failed. To address this, replication was introduced.

4.2 Design and Assumptions

- **Replication Strategy:** Data is replicated on RR successive nodes.
- **Fault Tolerance:** Replication manager re-replicates data upon node failure.
- **Consistency:** Periodic synchronization ensures replicas remain consistent.

4.3 Challenges and Solutions

Challenge	Solution
Data consistency across replicas	Used a synchronization mechanism for periodic updates.
Node churn causing instability	Stabilization was enhanced to handle churn dynamically.
Balancing replication overhead	Limited replication factor to maintain efficiency.

4.4 Implementation

The replication manager was added to the base system. Data was replicated to RR nodes upon insertion. On node failure, replicas were redistributed dynamically.

---

5. Benchmarking

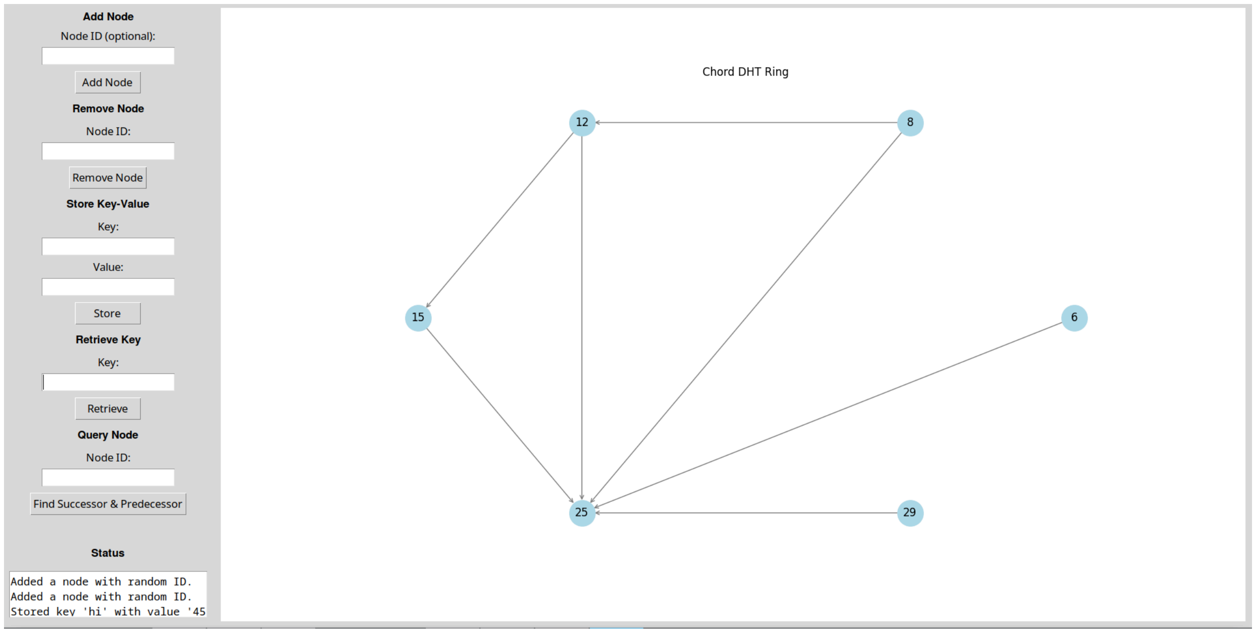
5.1 Setup

- **Simulation Parameters:**$M = 6\ (64\ slots),\ R = 3.$
- **Metrics Evaluated:**
  - Lookup efficiency $(O(logN))$.
  - Fault tolerance under node failures.
  - Scalability with increasing nodes.

5.2 Results

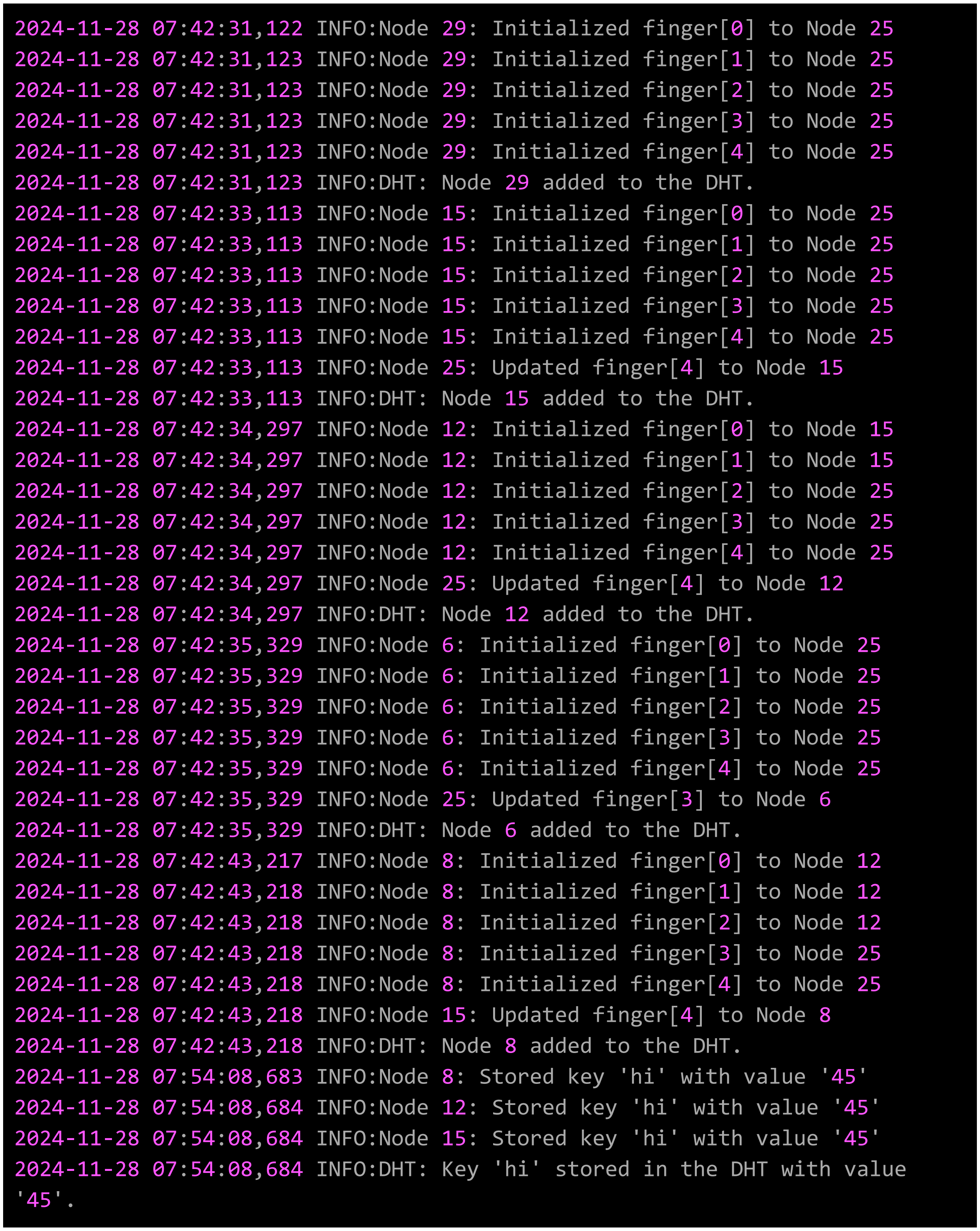
- **Lookup Efficiency:** Maintained $O(logN)$hops for lookup.
- **Fault Tolerance:** Data availability persisted despite 30% node failure.
- **Scalability:** Performed well with up to 64 nodes.

5.3 Sample visualisation



```
Added a node with random ID.
Added a node with random ID.
Stored key 'hi' with value '45
```

```
Matplotlib is using the 'TkAgg' backend.
2024-11-28 07:42:29,609 INFO:Node 25: Joined as the only node in the DHT.
2024-11-28 07:42:29,609 INFO:DHT: Node 25 added to the DHT.
2024-11-28 07:42:30,609 INFO:Node 25: Predecessor updated to Node 25
```



```
2024-11-28 07:42:31,122 INFO:Node 29: Initialized finger[0] to Node 25
2024-11-28 07:42:31,123 INFO:Node 29: Initialized finger[1] to Node 25
2024-11-28 07:42:31,123 INFO:Node 29: Initialized finger[2] to Node 25
2024-11-28 07:42:31,123 INFO:Node 29: Initialized finger[3] to Node 25
2024-11-28 07:42:31,123 INFO:Node 29: Initialized finger[4] to Node 25
2024-11-28 07:42:31,123 INFO:DHT: Node 29 added to the DHT.
2024-11-28 07:42:33,113 INFO:Node 15: Initialized finger[0] to Node 25
2024-11-28 07:42:33,113 INFO:Node 15: Initialized finger[1] to Node 25
2024-11-28 07:42:33,113 INFO:Node 15: Initialized finger[2] to Node 25
2024-11-28 07:42:33,113 INFO:Node 15: Initialized finger[3] to Node 25
2024-11-28 07:42:33,113 INFO:Node 15: Initialized finger[4] to Node 25
2024-11-28 07:42:33,113 INFO:Node 25: Updated finger[4] to Node 15
2024-11-28 07:42:33,113 INFO:DHT: Node 15 added to the DHT.
2024-11-28 07:42:34,297 INFO:Node 12: Initialized finger[0] to Node 15
2024-11-28 07:42:34,297 INFO:Node 12: Initialized finger[1] to Node 15
2024-11-28 07:42:34,297 INFO:Node 12: Initialized finger[2] to Node 25
2024-11-28 07:42:34,297 INFO:Node 12: Initialized finger[3] to Node 25
2024-11-28 07:42:34,297 INFO:Node 12: Initialized finger[4] to Node 25
2024-11-28 07:42:34,297 INFO:Node 25: Updated finger[4] to Node 12
2024-11-28 07:42:34,297 INFO:DHT: Node 12 added to the DHT.
2024-11-28 07:42:35,329 INFO:Node 6: Initialized finger[0] to Node 25
2024-11-28 07:42:35,329 INFO:Node 6: Initialized finger[1] to Node 25
2024-11-28 07:42:35,329 INFO:Node 6: Initialized finger[2] to Node 25
2024-11-28 07:42:35,329 INFO:Node 6: Initialized finger[3] to Node 25
2024-11-28 07:42:35,329 INFO:Node 6: Initialized finger[4] to Node 25
2024-11-28 07:42:35,329 INFO:Node 25: Updated finger[3] to Node 6
2024-11-28 07:42:35,329 INFO:DHT: Node 6 added to the DHT.
2024-11-28 07:42:43,217 INFO:Node 8: Initialized finger[0] to Node 12
2024-11-28 07:42:43,218 INFO:Node 8: Initialized finger[1] to Node 12
2024-11-28 07:42:43,218 INFO:Node 8: Initialized finger[2] to Node 12
2024-11-28 07:42:43,218 INFO:Node 8: Initialized finger[3] to Node 25
2024-11-28 07:42:43,218 INFO:Node 8: Initialized finger[4] to Node 25
2024-11-28 07:42:43,218 INFO:Node 15: Updated finger[4] to Node 8
2024-11-28 07:42:43,218 INFO:DHT: Node 8 added to the DHT.
2024-11-28 07:54:08,683 INFO:Node 8: Stored key 'hi' with value '45'
2024-11-28 07:54:08,684 INFO:Node 12: Stored key 'hi' with value '45'
2024-11-28 07:54:08,684 INFO:Node 15: Stored key 'hi' with value '45'
2024-11-28 07:54:08,684 INFO:DHT: Key 'hi' stored in the DHT with value
'45'.
```

---

6. Work Split and Other Details

6.1 Work Split

Task	**Team Member(s)**
Base Implementation	Shiva Shankar
Replication Mechanism	Sachin Hegde
Performance Benchmarking	Shiva Shankar
Documentation and Visualization	Sachin Hegde

6.2 References

1. Chord: A Scalable Peer-to-peer Lookup Protocol for Internet Applications.
2. Research papers on consistent hashing and DHTs.
3. Python Libraries: NetworkX, Matplotlib.

6.3 Links

- Chord Protocol Documentation
- Replication Strategies in Distributed Systems

---

7. Conclusion and Future Work

7.1 Conclusion

The project successfully implemented a scalable Chord DHT with replication. It demonstrated:

- Efficient data lookup $O(logN)$.
- Fault tolerance even with significant node failures.
- Scalability under dynamic conditions.

7.2 Future Work

- **Dynamic Replication Factor:** Adjust replication dynamically based on node reliability.
- **Load Balancing:** Prevent data hotspots in specific nodes.
- **Enhanced Failure Detection:** Use active heartbeats to identify failed nodes promptly.

This project provides a robust foundation for future research in distributed storage systems.