Report 5: Chordy – A Distributed Hash Table

Lakshmi Srinidh Pachabotla

October 9, 2024

1. Introduction

This homework assignment mainly covers implementing a distributed hash table (DHT) using the Chord protocol which provides efficient lookup and storage of key-value pairs. The main purpose of this is to construct a ring of nodes, add and lookup keys, handle node joins and departures, and ensure fault tolerance through key replication. It also validates its dynamic behavior in dynamic environments, such as node failures and new node additions.

By arranging the nodes in a virtual circular identifier space, the chord technique makes it possible for the node to swiftly recognize the node in charge of a given key. The stability, routing, and fault tolerance of the chord system are evaluated in this assignment.

2. Main Problems and Solution

There were a few problems faced while implementing. These problems were addressed separately and solved.

- The major problem is in the understanding of this assignment. It took some time to understand and go through the homework. It is possible only by investing more time and work in it.
- The second problem is with the ring construction and stabilization, in which the chord ring must be properly constructed with each node knowing its correct predecessor and successor. If stabilization does not occur, nodes may not be able to forward messages correctly, leading to lookup failures. This is achieved with the help of a stabilization process to ensure each node correctly identifies its successor and predecessor. During node joins and stabilization, nodes exchange messages to notify others of their presence and update their routing information. This is done with the help of "`timer:sleep(2000)`" which will allow time for the stabilization process to be completed before going to the new operations.

- The next problem faced while implementing is about the key addition and lookup, in which the system must be capable of routing lookup requests via the ring and keys must be placed in the appropriate node according to the node's identity. The system will not work as a distributed hash table if keys are not inserted or searched up appropriately. To counter this, keys were added and lookup using the "`test:add/3`" and "`test:lookup/2`". The system was verified by adding the key "123" with the value "Hello" and checking it against several nodes. Logging and debugging output contributed to ensuring that communications were routed correctly.
- The system must make sure that when additional nodes enter the ring, they take over the part of the key range from the already joined nodes. If it fails to give the correct keys, it could result in inconsistencies in key ownership. The system made sure Node 4 occupied the correct area of the key space when it was added to the ring. The key handover procedure enabled the new node to assume ownership of keys within its range, and stabilization was utilized to guarantee that the new node accurately recognized both its predecessor and successor.
- The next problem faced while implementing is the handling of node failure and replication. During the execution process, the system must be able to recover by stabilizing the ring again and ensuring that any keys previously stored on the failed node are still accessible. If the keys are lost due to node failures, the system will lose fault tolerance. To overcome this situation, key replication was implemented, in which keys are replicated across multiple nodes so that if one node fails, another node still holds the replicated keys.

3. Evaluation

The system was evaluated by performing the following operations:

- **Ring Construction**: Started four nodes and verified that they formed a stable ring. Each node knew its predecessor and successor, ensuring proper routing within the ring.

```
1> Node1 = test:start(node3).
<0.87.0>
```

```
2> Node2 = test:start(node3, Node1).

<0.90.0>

3> Node3 = test:start(node3, Node2).

<0.93.0>

4> timer:sleep(2000).

ok
```

- **Key Addition and Lookup**: The key "123" with the value "Hello" was added through Node1 and successfully looked up from Node2. This demonstrates that the system correctly routed the lookup request to the node responsible for the key.

```
5> test:add(123, "HELLO", Node1).

ok

6> Value = test:lookup(123, Node2).

{123,"HELLO"}
```

- **Node Join**: After adding Node4, the ring continued to function correctly, and the new node properly joined the ring. The system ensured that key responsibility was transferred, maintaining consistency in key distribution.

```
7> Node4 = test:start(node3, Node3).

<0.99.0>

8> timer:sleep(2000).

ok
```

- **Node Failure Handling**: After stopping Node2, the system re-stabilizes, and a lookup for the key "123" from Node3 returned the correct value, demonstrating that the system maintained key availability through replication.

```
9> Node2 ! stop.

Successor of #Ref<0.2469904816.92012547.46623> died

Stop
```

```
10> timer:sleep(2000).

ok

11> Node1 ! probe.

Probe time: 102 micro seconds

 Nodes: [945816365,501490715,443584618]probe

12> Value = test:lookup(123, Node3).

{123,"HELLO"}
```

- **Probe Mechanism**: Probes were sent from Node1 both before and after Node2 was stopped. The system reported that the probe was completed successfully, indicating that the ring remained intact even after the failure of a node.

4. Conclusions

This Chord implementation effectively fulfilled the following goals such as, the stabilization was made to guarantee that every node understood its neighbours and that the ring was built appropriately. Key addition and lookup functioned as anticipated, and messages were routed between nodes appropriately. Node joins and failures were managed without incident, and following the Node2 failure was recovered and continued to function by the system. Fault tolerance was guaranteed via the key replication system, which made sure that keys were accessible even in the event of node failure. Overall, it was tough to execute, and took so much time to understand and go through what was happening. This homework uses the chord protocol to illustrate the essential components of a strong distributed hash table. Because of its fault tolerance and ability to adapt to dynamic network changes. This system is appropriate for distributed systems where nodes may add, exit, or malfunction at any moment.