Chordy: a distributed hash table

Nikhil Dattatreya Nadig

October 12, 2017

1 Introduction

The objective of this exercise was to implement a distributed hash table following the Chord scheme. The distributed hash table is designed in a virtual ring structure so that nodes are able to easily join and leave. A distributed hash table (DHT) is a class of a decentralized distributed system that provides a lookup service similar to a hash table: (key, value) pairs are stored in a DHT, and any participating node can efficiently retrieve the value associated with a given key. Responsibility for maintaining the mapping from keys to values is distributed among the nodes, in such a way that a change in the set of participants causes a minimal amount of disruption. This allows a DHT to scale to extremely large numbers of nodes and to handle continual node arrivals, departures, and failures.

Features of Distributed Hash Tables:

- Fault Tolerance - The rings are capable of handling nodes leaving and joining.
- Uniform Distribution - All the key-value pairs should be evenly distributed. This way no node is overburdened with excess data.
- Data Replication - When nodes are lost, it is important to retain the data that was previously stored by the lost node.

2 Main problems and solutions

The distributed hash table is built in two iterations. First, a DHT's ring structure is created without storage capabilities. In DHT, each node keeps track of it's successor and predecessor.When all the nodes in the table keep track of their successors and predecessor, a ring structure is formed. This way, if a node needs to be added or removed from the table, only the nodes before and after need to update the key values to retain a ring structure.

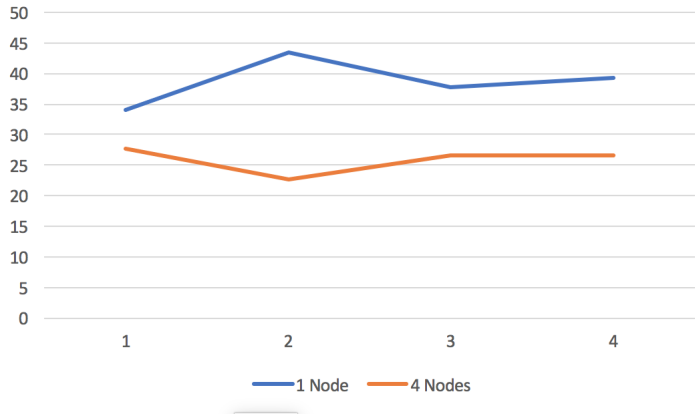

Figure 1: time comparison for 5000 look ups with 1 node vs 4 nodes.

A periodic stabilize procedure will consist of a node sending a *request* message to its successor and then expecting a *status* in return. When it knows the predecessor of its successor it can check if the ring is stable or if the successor needs to be notifies about its existence through a *notify* message. In the second iteration, we add a storage mechanism to the distributed hash table. To add a new key value we must first determine if our node is the node that should take care of the key. A node will take care of all keys from (but not including) the identifier of its predecessor to (and including) the identifier of itself. To handle this, we create a lookup function which determines if a given node is responsible for the new node or not.

3 Evaluation

In the first version of the solution, we create a ring structure with distributed hash tables. In the second, we add a data structure so that we can store data. The figure 1 depicts the time difference in time to look up 5000 keys with 1 node in the distributed hash table and 4 nodes. The single node table took on an average 38.5 milliseconds to look up a table with 5000 keys. Whereas, a table with 4 nodes took 25.9 milliseconds to look up the same amount of keys.

4 Conclusions

We have understood the importance of distributed hash tables and their uses in a distributed environment.