Chordy

Vasigaran Senthilkumar

October 13, 2021

1 Introduction

A distributed hash table algorithm similar to the chord protocol is discussed in this assignment. The chord protocol which was developed at MIT is a peer to peer protocol which offers fault tolerance and scalability. In this assignment a network of nodes were implemented with each node storing only the data it is responsible for.

2 Main problems and solutions

Reading the chord paper was really interesting but when it came to implementing there were confusions about stabilizing the network and determining predecessors and the successors. I had to refer to previous year lecture of DHT and chord protocol to get a proper understanding.

2.1 The breakdown of stabilising

Node A and Node B are connected in a network where Node A is predecessor and successor to Node B. Node C joins the network to assigns Node B as its successor. During the scheduled stabilise Node C runs the stabilise. node C requests its successors Node B to reply its predecessor. if the predecessor is Null or itself then Node C notifies Node B That it can be the predecessor.If Predecessor is other node A, we check that if that Node A is between node C and Node B using key:between(). It is important to note that the network is in a ring structure. Now Node A is a better successor to node C so Node C requests Node A to be the successor to repeat the stabilisation again by setting it as the successor. The mistake I was making at the beginning in the stabilising was not calling the stabilisation again with a request to a new successor.

2.2 Implementation of Storage

The lecture was very much helpful implementation of the storage. The node responsible for the Key,Value is identified by almost the same functionality

implemented in earlier section. If new key is between the predecessor's key and its own key, then the new key and value can be stored in its storage. If that is not the case then the new key and value is sent to its successor to check there. The exact same functionality is used in lookup of a key. The tricky part of the storage was about joining of new nodes and then distributing the responsible keys to the new nodes if any. We use the handover function to check for the keys that new incoming node is responsible for assign it to it. The Rest is Kept as it is. The split and merge functions are used for this purpose. I was facing issue at this point by getting wrong key value pairs in wrong nodes. Then I figured that Partitions function I used to split into two lists, the first list contains all elements for which Pred(Elem) returns true, and the second list contains all elements for which Pred(Elem) returns false. I had returned vice versa and corrected it at a later stage

2.2.1 pseudocode for storage

```
key(Id){\subset}{(Pred(Id),Id]
```

3 Evaluation

As part of evaluation to check if node1 is implemented in a right way. A probe was introduced to trace the ring of network. When the probe message was sent, the call is recursively passed on to the successors until it reaches the same node again. The time it took to pass around the node is also taken note of. The time to probe the ring increased with increase in the number of nodes as expected. The result is plotted in Figure 1

Then I had a test case to check if a new node is introduced it takes responsibility of its keys in the store. I started 3 nodes P1,P2,P3 with id 1,5, and 10 respectively. Then added a key value pair with id 3. Thus it went to the storage of P2. Now I introduced a new node P4 with id 4, Then key 3 moved to P4 and was removed from node P2. The screenshot for the same is available in Figure 3.

Finally I tried to simulate the lookup time for 5000 messages for different number of nodes. With the increase in the number of nodes , the lookup time reduced to some extent. I also felt that an even distribution of keys among each node will result in better scalability. the results of this test case are plotted in the form of a graph.Then I had implemented an extra add multiple keys functionality which results in the time it takes to add 5000 key Value pairs. .

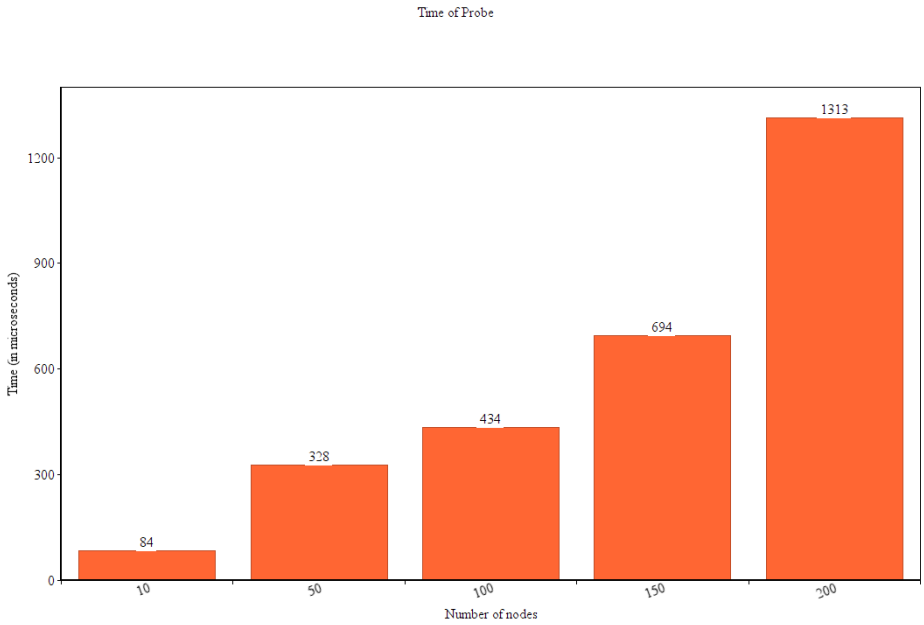

Figure 1: Probe Time

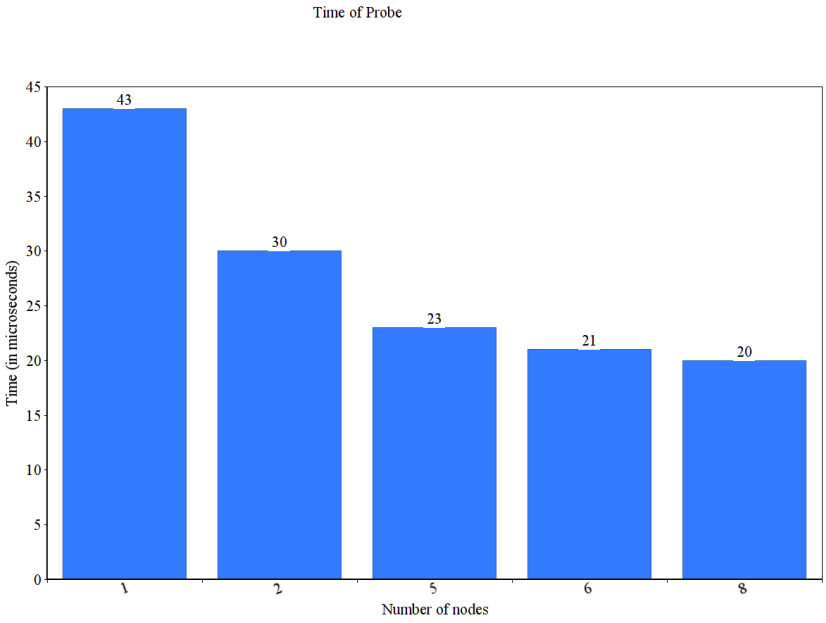

Figure 2: Lookup time for 5000 key values

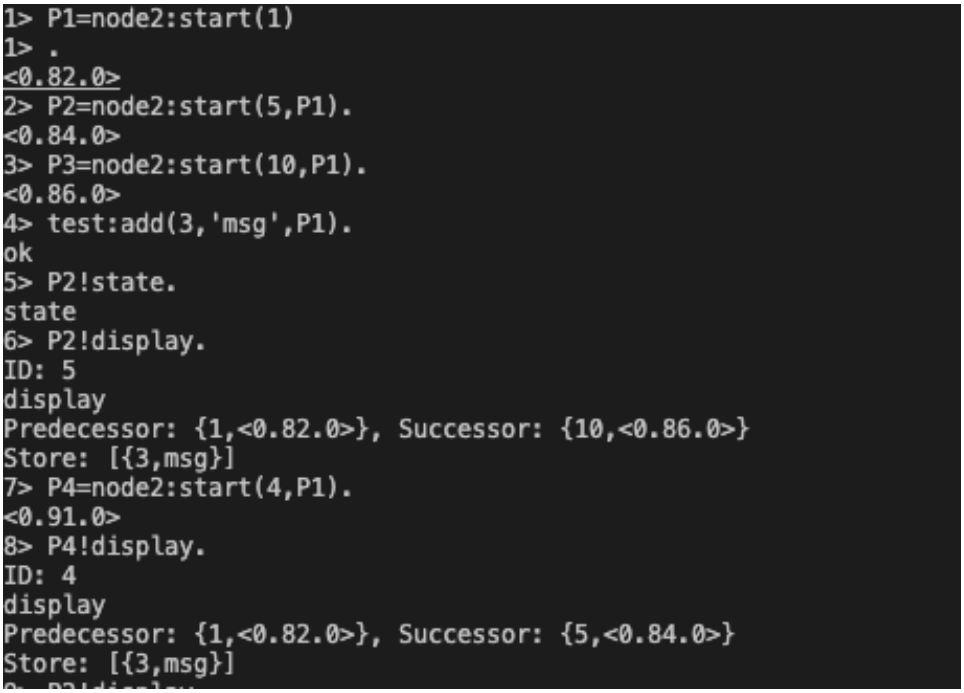

Figure 3: Handover testing

4 Conclusions

Thus the implementation and performance evaluation of chordy was done. The lectures were the best content material to work on this assignment. The working of distributed hash tables was understood in depth during the implementation. I also did the extra part of handling node failures, Where the successor's successor is also tracked. In case of successors failure the next becomes successor.