# Chordy: a distributed hash table

Mikael Karlsson

September 30, 2019

## 1 Introduction

The goal with this assignment was to implement a distributed hash table following the cord scheme. The assignment was separated into to two tasks, the first task consisted of constructing a ring structure with nodes where each node is assigned with a random key. In the second task a store was added to each node. The store consisted of a list with {key, value} pairs.

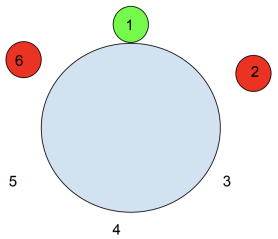
## 2 Main problems and solutions

The assignment was divided into to parts Node1 and Node2.

### 2.1 Node1

Most of the code was provided, so i spent some time to understand how things work. The hardest part was to implement a function called between and to understand how a network is updated when a new node wants to join. The purpose of the between function is to return true or false whether a number x is between a number y and z. This was kind of difficult because the implementation consist of a ring structure demonstrated in Figure 1.

Figure 1: Example of nodes connected in a ring structure




A network is updated with a function called stabilize. Each node will call the function with a fixed interval to set it's successor and predecessor. If we want to add the green node 1 to a network with the nodes 2 and 6 as visualized in figure 1.

1. 1 takes 2 as successor
2. 1 send a request to 2 to tells its predecessor.
3. 2 sends Predecessor 6.
4. check if 6 is between 1 and 2
5. answer is no, 1 notify 2 that he is the new predecessor
6. node 6 runs stabilize after some time and asks it's successors predecessor. 6 will get node 1, and the procedure is repeated.

### 2.2 Node2

In the second task we implemented a storage (hash-table) to the already working ring structure. Each node is responsible of a set of key's. If we have a network of 3 nodes 1 ,2 and 6 as in Figure 1. Then will node 1 be responsible for key: 1, node 2 will be responsible for key:2 and node 6 will be responsible for key: 6,5,4,3.

## 3 Evaluation

I evaluated the first task by creating a network of 5 nodes and sent a probe message to see that everything worked. The result was:

```
Tid= 1569870803240484
Noder= [10840645,939340508,897916454,480798909,287249153]
```

I evaluated node2 by adding 4000 keys to network and then checked how long it took to do 4000 lookups. I tested with 6 different network setups. The result was:

```
1 nodes:
4000 lookup operation in 23 ms
0 lookups failed, 0 caused a timeout

2 nodes:
4000 lookup operation in 17 ms
0 lookups failed, 0 caused a timeout

4 nodes:
4000 lookup operation in 17 ms
0 lookups failed, 0 caused a timeout
ok
```

```
8 nodes:
4000 lookup operation in 16 ms
0 lookups failed, 0 caused a timeout
ok

40 nodes:
4000 lookup operation in 42 ms
570 lookups failed, 0 caused a timeout
ok
```

## 4 Conclusions

I think this task was interesting and it was cool too see how it is possible to maintain a distributed hash table with a ring structure. The idea with cord scheme was relatively easy to understand but pretty hard to implement.