# Chordy — A distributed hash table

Anton Bothin

October 9, 2019

## 1 Introduction

The aim of this assignment was to implement a distributed hash table following the Chord scheme. In Chord the nodes are structured in a ring and we have procedures to store and lookup key-value pairs. In our implementation failure handling is limited to one successor and all data is replicated once so that it is not lost when a node dies.

## 2 Main problems, solutions, and evaluation

### 2.1 Building a ring

To keep nodes in a ring structure we use two pointers, one to the nodes predecessor and the other to its succesor, creating a circular doubly linked list. Maintaining the ring structure is done by stabilizing at set time intervals. To stabilize, a node will send {`request, self`} to its successor and will get the succesors predecessor in return, let's call this `Pred`. If `Pred` is `nil` or if it is the same as the succesor (pointing to itself) the node should suggest becoming the new predescessor. If `Pred` is pointing to another node we need to check if our position is inbetween the nodes or not. If we are not inbetween `Pred` and our successor we adopt that node as our successor and send {`request, self`} to it, causing it to reply with its predecessor and initialize the stabilization again. Otherwise we should be inbetween the nodes by suggesting ourself as predecessor to our the successor. When the succesor receives this suggestion it will itself check that the suggested predecessor is a better fit than the current one.

### 2.2 Adding a store

Adding and looking up values is done by first finding the responsible node. We do this by checking if the key is inbetween the current nodes predecessor id and its own id. If it is, the key-value pair should be stored in the node. Otherwise the request should be forwarded to its successor. When a new

node joins we should hand over the values with keys lower than the new predecessors key.

When testing the performance of our distributed hash table I only had access to a single machine. The testing I did was adding and looking up 4000 values. I here noticed that it did in fact go quicker when using several nodes (distributing the hash table) compared to a single node. The difference was however not significant, adding a node only decreased the time by about one tenth of a second. Since I did the tests locally on a single machine it may be the case that the time would not differ had I done it with several machines where network traffic slows down the transfer rate of messages between nodes. My conclusion is however still that it is the searching of values that is the bottleneck, while forwarding messages is quick in comparison.

### 2.3 Handling failures

The only type of failure we handle is when nodes go down. We do this by keeping a pointer to the successor of our successor. To detect nodes going down we use the built in monitor functions. Each node monitors both its predecessor and succesor. When a node receives a 'DOWN' message it first have to find out if it is the predecessor or succesor that went down. In the case of predecessor we simple set the nodes new predecessor to `nil` since the actual predecessor will eventually suggest itself when the stabilize procedure is run. If the succesor is the one that went down we simple adopt the next succesor as our new succesor.

We still have the problem of false detection, if a node is temporarily down it might be removed from the ring by the other nodes. If the node that went down still has the correct references to its predecessor and succesor when it goes back up it will eventually be included in the ring again by the stabilize procedure. If the node has lost its references (maybe because it thought that those nodes went down as well) it won't be able to reconnect to the ring.

## 3 Conclusions

This assignment took quite a lot of time since I had to implement a lot of the code myself, unlike the previous assignments where almost all code was given. Because of this I gained a really good understanding of how the Chord scheme works and how to implement a distributed hash table. I also gained knowledge of the problems that exists and how to tackle them to make the system fault tolerant.