# Report 5: Chordy

Nikita Gureev

October 4, 2016

## 1 Introduction

In this homework a distributed hash table following the Chord scheme is implemented. The implementation should be able to maintain a ring structure with the possible addition and removal of nodes. It also should be able to store {Key, Value} pairs and be able to look them up. The aim is to have a ring of nodes that store data that can be looked up.

## 2 Main problems and solutions

The main problems that were encountered during this task were the implementation of successor and predecessor changes, handling the failure of a node. The first problem was present as the incomprehension of algorithm itself and not enough understanding of source code. The second problem was optional and involved keeping track of nodes and handling their failure. It was easier to a degree, as it was already encountered and solved in previous tasks.

## 3 Evaluation

The first problem was solved through reading the original paper on Chordy and understanding the mechanics of how the ring structure was maintained during node addition or removal. After the algorithm was understood in enough detail, the implementation itself was not as difficult, but the purpose of different messages was still unclear. It took some time to comprehend which messages should be sent in which case. The chosen number of successor node, specifically one, is enough in the current context, as the number of nodes is small. With a growing number of node it is sensible to have more than one backup successor node, however with the increased reliability the performance decreases.
During the implementation some tests were conducted, and performace measured.

| | Trial 1 | Trial 2 | Trial 3 | Trial 4 | Average |
|---|---|---|---|---|---|
| Time to process 4000 requests from a single node, ms | 31 | 31 | 31 | 31 | 31 |
| Time to process 4000 requests from 4 concurrent sources, ms | 44 | 30 | 45 | 31 | 37.5 |

Table 1: Benchmark tests

The average time to process 4000 requests from four different sources was around 20% greater than the processing time for a single source. As in both cases the requests are processed sequentially, the only difference that is introduced in the second case is the increased overhead. The second problem was more also quite challenging as it included additional handling of failures during execution. However, the fact that it was not different in how the state of successor nodes is monitored from the previous assignments made it considerably easier to implement. The fact that the algorithm for maintaining the ring structure already had the change of successor and predecessor also eased the understandin and implementation of the solution to this problem.

# 4 Conclusions

In conclusion, this assignment provides a number of essential insights into how distributed hash tables are implemented, how they maintain the structure and consistency in case of node addition, removal, and failure. This has great importance, as the hash tables are one of the most widely used datasructures, and their distributed implementation is also ubiquitous in the distributed systems. The underlying mechanisms of ring strucutre maintenance, failure handling, and data replication enable students to grasp how such real-world systems operate.