Homework 5 Report: Chordy

Justas Dautaras

October 4, 2019

1 Introduction

In the fifth and final Distributed Systems task a distributed hash table is implemented, following the Chord scheme. The nodes are connected in a ring architecture, where each one of them has a successor and a predecessor. And in this way, all the data is distributed between these nodes.

This task is divided into couple main modules:

· **Key** This module is used to generate random keys and check if a key is in a specified interval.

· **Node** This is the main module of this task. It allows nodes to connect and communicate to keep the ring correct and working.

· **Storage** This module is used to handle the main lookup function and to manage the distribution of data across nodes.

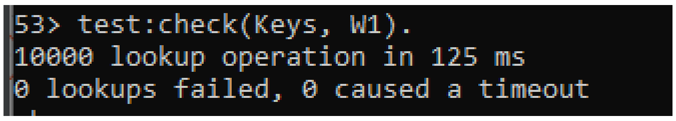
Completion of this task allows to test distributed hash tables efficiency and to understand how it works.

2 Main problems and solutions

One of the main problems in central systems that this task explains is efficiency. Where distributed systems in most cases are a lot more efficient. In this paper we are going to evaluate the difference in performance with a single node and multiple nodes with information distributed between them. This will be done by performing some tests described below.

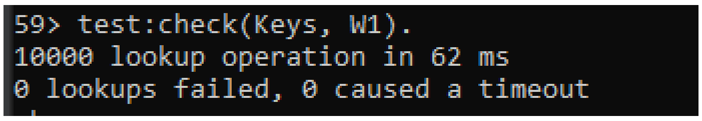
3 Evaluation

To test this, we are going to measure the time it takes to lookup 10.000 items in a single node system, in 10, 100 nodes system. And what if for the last case we had four machines search 2500 items each? Well, the time would just be about four times lower. Test results:



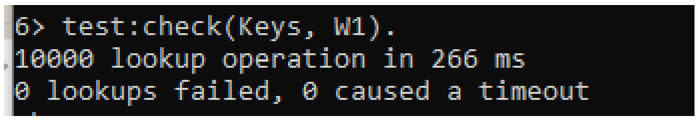
```
53> test:check(Keys, W1).
10000 lookup operation in 125 ms
0 lookups failed, 0 caused a timeout
```

Figure 1: 10.000 items in a single node system



```
59> test:check(Keys, W1).
10000 lookup operation in 62 ms
0 lookups failed, 0 caused a timeout
```

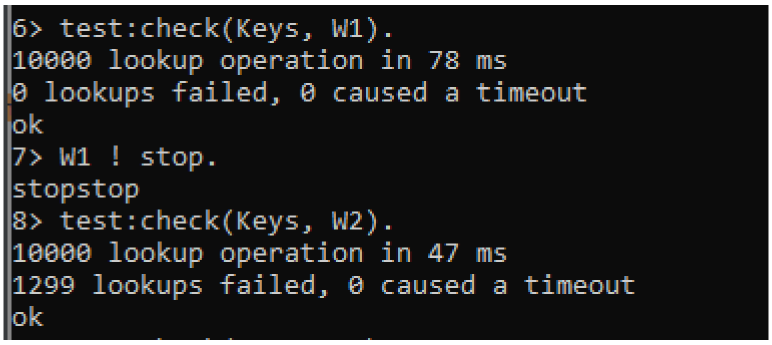
Figure 2: 10.000 items in a 10 node system



```
6> test:check(Keys, W1).
10000 lookup operation in 266 ms
0 lookups failed, 0 caused a timeout
```

Figure 3: 10.000 items in a 100 node system

Also, having implemented the additional failure detection option, we can see how a dead node takes away the items It was holding in the last picture. More has to be done for the information to be kept safe.



```
6> test:check(Keys, W1).
10000 lookup operation in 78 ms
0 lookups failed, 0 caused a timeout
ok
7> W1 ! stop.
stopstop
8> test:check(Keys, W2).
10000 lookup operation in 47 ms
1299 lookups failed, 0 caused a timeout
ok
```

Figure 4: System with failure detection

4 Conclusions

As we can see, distributed hashing tables are efficient only up until some point. Too many nodes might make the seach not as efficient, since every single node has to be checked to find the needed element. Although, what this solutions also adds, is safety, since every single node has to crash to lose the information.

Solving this task has taught me to program on a more intermediate level of functional programming and helped to further extend my knowledge on the fundamentals of distributed systems.