Report 5: Chordy

Alan Goran

October 12, 2017

1 Introduction

A major problem in modern applications is to efficiently look up a node that stores a particular information, given that each node is a different machine. And here is where the chord protocol of the distributed system comes in handy. This protocol is a vital solution for the efficient communication issues nowadays because of the extreme demand of information.

The task of this assignment was to implement a distributed hash table following a chord protocol. The hash table is organized as a ring structure where we are able to add and look up nodes. The ring contains different types of nodes, i.e. successors and predecessors, where each node will keep track of its successor and predecessor to be able to repair the ring in case of failures.

2 Main problems and solutions

The main problems of this assignment were two things, implementing a ring structure and adding a storage.

2.1 Ring structure

The first request we had on our implementation was to be able to handle a growing ring. We need therefore unique keys for the objects in order to keep track of them. Instead of using names to create unique keys for the hashing as the chord protocol suggests, we used randomly generated numbers as keys when they were added. We simply hoped that the randomly generated number for a new key that is joining the network is unique. Furthermore, each node is responsibly for a range of keys and each node is aware of the nodes that are its successor and predecessor to make sure that they were handling the correct keys continuously. A vital problem with the key was placing it in the right place. It is easy to forget that we are dealing with a ring, so if we have a ring that goes from 1 to 20 a number of value 2 can

be between 5 and 20. Therefore, a feature that handled this had to be added.

Furthermore, when the ring is constructed and a new node is about to join the ring then stabilization of the ring is needed. This is in order to make sure that all the nodes have the right successor and predecessor.

Lastly, a probe function was implemented. The probe function visited all the nodes and circles around the ring until it returned to the node with the same processor ID as the node from where it was sent. This was used as a test to know whether the ring is complete and correct.

2.2 Adding a storage

Every node is responsible for a specific amount of information to store. Each information in a given node have a key that corresponds to its identity in the ring. The storage works by using Key-Value items. This means that by calling the key you can access the value which is the information we seek.

Each node could now receive two more messages; add and lookup. The two functions stand for adding items and looking up an already existing item. Both functions work quite similarly, first it checks if the node is the right node to add or to look up something, if yes then it will execute the command and if not then pass it forward.

Another problem that needed to be solved was when a new node joined the ring it caused an interruption between the nodes. This was solved by the function handover, which simply makes the new node in charge of some of the storage that was previously handled by its neighbour node.

3 Evaluation

There were several tests that were done on this implementation. The first one was to check the difference between handling 4000 elements on one machine compared to 4 machines that handles 1000 elements each. This was done by dividing the 4000 elements (keys) on 4 nodes on 4 machines, where node one (N1) was responsible for keys from 1 to 1000, N2 responsible for keys from 1001 to 2000, N3 from 2001 to 3000 and N4 from 3001 to 4000. For this particular test, the nodes and the keys were manually set to these values and not done by randomly distributing it. The results of this test showed that it takes much longer to divide the elements on different machines compared to one machine handling everything on its own. The downside of doing this only on 1 machine is although that it has to store all the elements on its own.
The second test was to implement a test procedure that adds a number of

random key-value pairs into the system and keeps the keys in a list. Then, a lookup of all these keys were done and the time it took for the lookup was measured. The result for this experiment is listed below.

```
# The test below is done on one machine only

ONE NODE:
4000 lookup operation in 39 ms
0 lookups failed, 0 caused a timeout
ok

TWO NODES:
4000 lookup operation in 27 ms
0 lookups failed, 0 caused a timeout
ok

THREE NODES:
4000 lookup operation in 26 ms
0 lookups failed, 0 caused a timeout
ok

1000 NODES:
4000 lookup operation in 1073 ms
```

So there is not a sizable difference between 1,2 and 3 nodes but a major difference when 1000 nodes is added and the same check is done.

4 Conclusions

This assignment was the most interesting assignment in this course. It gave a major understanding of the Chordy protocol and a great insight on how powerful distributed systems can be.

Doing this task made me realize that information data can be stored on many different machines and still be accessed elsewhere in an efficient way. This gives a great insight of how we can solve the major demands of data storage that we face every now and then.