Homework 5 Chordy - a distributed hash table

1 Introduction

The assignment was to implement a distributed hash table following the Chord scheme. A distributed hash table stores key-value pairs by assigning keys to different nodes. A node will store the values for all the keys for which it is responsible. Hashing distributes the key-value pairs across nodes in a network. The nodes can retrieve a value given a key.

Chord specifies how keys are assigned to nodes and how a node can retrieve a value given a key by locating the node responsible for that key. The nodes are arranged in a ring structure where each node has a successor who is the next node in the ring and predecessor is the previous node. The nodes also has an ID.

The DHT will be designed as a ring goal is to build a network of nodes where each node has a key identifier.

2 Main problems and solutions

2.1 First implementation - a growing ring

The first implementation handles a growing ring, nodes are structured as a ring with a successor and a predecessor. There was a key module that generated a random number for the key and also checked if a Key was in between two other keys in the ring. We also had a node1 module where a stabilize function was used to maintain a ring structure when a new node wants to enter. The node sends a request message to its successor to get the predecessor of its successor. Depending on the value of Pred, the node could suggest to be the new predecessor of the successor. The successor will receive the notify message of the suggestion from the node, call the notify function and decide if the node should be the new predecessor or not.

A probe message was introduced to check if the ring is actually connected:
{probe, I, Nodes, Time}
The probe message is forwarded around the ring until it reaches the node from the beginning and the time that it took to pass it around is measured.

2.1 Second implementation - add a storage

In the second version of Chordy, a local store to each node and the possibility to add and search for key-value pairs are added. An node's local storage contains all the nodes in between the predecessor of the node and the node itself. We have a handover function that delegates the storage to the nodes. We receive a request for a key that we want to add to

the storage. If the key we want to add lies between our node and the predecessor of our node, we add it to our node's local storage. If the requested key does not lie in between we forward the add request to the successor of our node instead. The same thing is done for a lookup request, when we want to look up a key in the storage.

2.3 Third implementation - handling failures

In this implementation we add support for node failure. We keep a pointer to the successor of our successor called Next. To detect if a node has failed we use the monitor function for a new predecessor or successor that sends a 'DOWN' message when a node has failed. If a predecessor has died we simply set it to nil and it will later receive a request of a possible new predecessor from the stabilize function. If our successor dies we will use Next as our new successor.

3 Evaluation

3.1 Probe message

When we send a probe message we get the following printout:

```
Nodes: [311326755,945816365,723040206,597447525,501490715,443584618]
probe
Time: 61 micro seconds
```

It shows that it took 61 micro seconds to pass around a probe messages in the ring of nodes. The more nodes you add to the ring the longer it will take for the probe message to be forwarded to all of them.

3.2 Local storage

To test a node's local storage, we can add for example 1000 nodes in the ring and when I did 10 lookup operations I got the following:

```
10 lookup operation in 7 ms
0 lookups failed, 0 caused a timeout
```

3.3 Testing failure

A kill function was added to the benchmark in order to be able to kill a node. A number of nodes was created and then a node was killed. A probe message was then sent to see if we would still receive an answer, which we did. This means that the probe message managed to be passed around in a ring and we can therefore conclude that the circle is complete and the support for failure was successful.

4 Conclusions

I found this assignment a bit tricky since there were several things that were needed to keep track of a node's successor and predecessor. It is the first time I have been introduced to the Chord scheme and it was interesting to read about. I also learned about how a distributed hash table works and actions that can be taken to make it fault tolerant if nodes should fail.