Chordy - a distributed hash table

Abdullah.
aabdull@kth.se

October 13, 2021

1 Introduction

The assignment was to implement a distributed hash table following the Chord scheme.

The objective of the assignment was to understand distributed hash tables and Chord Alogrithm and how a simple implementation of Chord Alogrithm in Erlang can be achieved.

2 Initial Implementation - Building a ring

Nodes in this implementation should always be structured as a ring with the help of two pointers, the successor and predecessor pointer. To maintain the ring structure a procedure stabilize(Pred, Id, Successor) was implemented. To stabilize a node will send request, self() to its successor and will get status, Pred in return, where Pred is the predescessor of its succesor. Depending on the value of Pred we should do the following:

- If nil the node should suggest adding itself as predescessor by sending notify, Id, self() to its successor.
- If it's pointing to itself same as nil.
- If it's pointing back to us everything is settled.
- If it's poiting to another node two points to be considered.

```
If Id is in between the predescessor of our successor
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.

stabilize(Pred,Id,{Xkey,Xpid})

Otherwise we should be in between the nodes by suggesting
itself as predecessor by sending {notify, {Id, self()}}
to the successor.
```

When a node receives a suggestion about a predecessor it will do some research before adopting it:

- If the node doesn't have any predecessor it adopts it right away.
- If the node have a predecessor it checks if the suggested predecessor is in between the current predecessor and itself. If it is the node will adopt the suggested predecessor and otherwise not.

We don't need to inform the new node about our decision since it will eventually issue a new stabilization and find out anyway because stabilization is done periodically.

3 Adding a Store

The crucial part in add and lookup methods was to check if the key is in between our predecessors Id and the current nodes Id. If it is, the key-value pair should be stored in the node, otherwise the request should be forwarded to its successor.

Another important point was when a new node joins the ring we should hand over the values exclusive of new predecessors key and inclusive of self key.

```
Exclusive and Inclusive notation

(newPredecessorskey, selfkey]
```

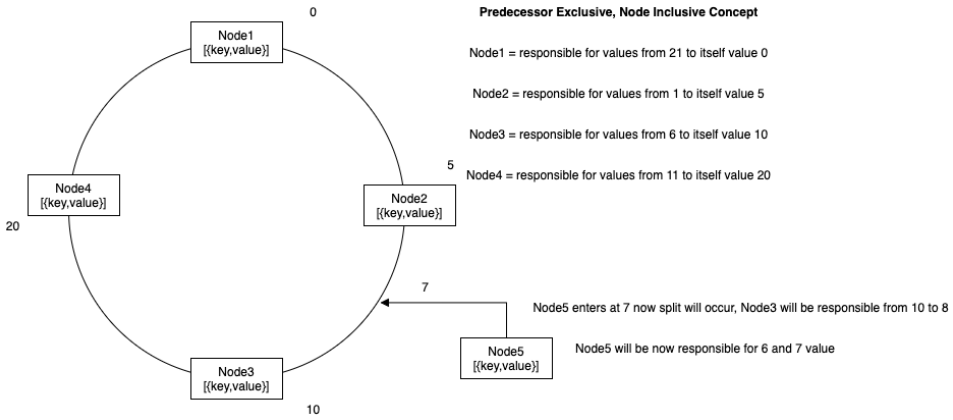

Figure 1: Chordy Architecture

4 Performance Evaluation / Results

Two tests were tried for performance evaluation

For first test only one node was created to handle 40k msgs for which lookup time taken is 30.81 seconds.

For second test four nodes were created to handle 40k msgs for which lookup time taken is 4.35 seconds

```
P = test:start(1,nil).
test:evaluate(40000,P).
Time Elapsed = 30810ms

P = test:start(4,nil).
test:evaluate(40000,P).
Time Elapsed = 4347ms
```

From the time difference it can be seen that the performance increases when values are distributed over many nodes in the ring structure.

5 Bonus Part - Handliing failures

To handle failure we keep a pointer to the successor of our successor. When a status message is sent the successor is also included and the stabilize function is now updating the next successor pointer when adopting a new successor. The next successor should then be set to the one we previously had as successor before adopting the new one.

To detect nodes going down we use the built in monitor functions. Every time we adopt a new successor or predecessor we should now monitor it and if we already were monitoring one when adopting the new one we should drop the old one first. When we receive a 'DOWN' message from the monitors we should find out if it's coming from our sucessor or predecessor. If its coming from the predecessor we do nothing but removing the reference to it since someone will eventually suggest a new one when running the stabilize function. If it's coming from the successor we should use our next successor as successor instead.

Although, we still have the problem of false detection and if a node is temporary down it might have been removed from the ring by the other nodes. If the node itself still has OK references to its successor and predecessor it will eventually run stabilize and be included in the ring again. It's worse if it thinks that those nodes are down at the same time as the next successor. Then it won't be able to reconnect to the ring by itself

6 Conclusions

This assignment helped me to understand how a Chord distributed hash table works and some of the problems we need to tackle to make it fault-tolerant.