1. Introduction

This assignment is about distributed hash tables (DHTs). A DHT functions as a regular hash table but is distributed in a network. This means that it is possible to store a {key, value} (pair) across the network, in a large number of different nodes. Consequently, given a key, one can efficiently look up a node’s stored value. The fundamental topology used in this assignment is a ring network topology. However, a ‘star-like’ topology could be created in order to reduce the time for traversing the network, which is necessary for a large network. (Coulouris, Dollimore & Kindberg, 2012)

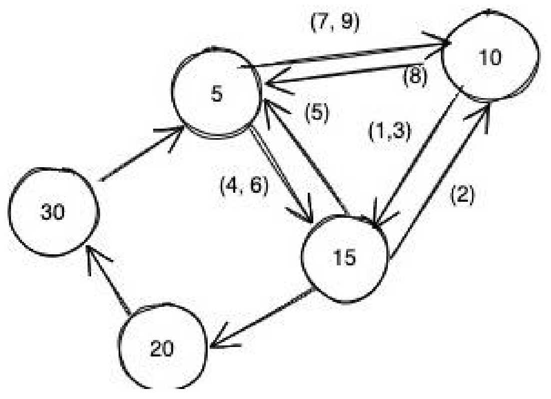
In a DHT, a new node can join or leave easily. For this, the ring needs to be *stabilized* in order to place the new node in between two others, in the correct place. This is done by comparing the Pid’s of nodes in the ring, until the new node’s predecessor and successor is found. Once found, the new node can enter the ring / network safely.

A probe is also introduced, in order to measure the time it takes to traverse the network. This is also done in order to check if the link is connected. Last but not least, storage is introduced in order to store the actual values.

Another thing that needs to be handled in a DHT is the occurence of a node crash. Therefore, fault tolerance needs to be considered.

2. Main problems occured

Stabilizing the ring was challenging. The hardest part was to interpret the given code and to figure out what code was missing. Also, which is not only the case for the stabilize/3 function, keeping track of and interpreting each variable was a mess. There are a lot of different ids, keys and peers being passed around. Going through the code over and over again, and creating the simple graph below was the solution for this, which was very time consuming.



Joining node (10): Who is your predecessor? (1)
Node in the table (15): It's at 5. (2)
Joining Node (10): Point to me then (3)
Node in the table (5); Who is your predecessor? (4)
Node in the table(15): It's at 10 (new node) (5)
Node in the table (5): Ok thats a good (6)
Node in the table(5): Who is your predecessor? (7)
New node(10): I don't have one! (8)
Node in the table(5): Use me! (9)

The solution of stabilization follows:

- If the predecessor is nil, if the ring is empty, the node can slide in. We are then notifying ourselves, and calling the node function again in order to wait for a message.
- If it is pointing to us (Id), nothing happens and we keep its successor by returning it.
- If it is pointing to itself, the same procedure as the ‘nil’ case is happening.
- If someone is entering {Xkey, Xpid), we need to check whether the joining node’s key is in between the predecessor (Id) and its successor (Skey).
    - True = we can adopt the joining node as our successor
    - False = same procedure as the ‘nil’ case is happening

3. Evaluation / Results

Testing on multiple machines was not possible and measuring differences in one. But the idea is that by running the DHT on different machines, it is obviously possible to handle each requested operation simultaneously, and therefore the time for each operation will be reduced.

Commands for starting and displaying the probe, adding and looking up values in the DHT will be demonstrated during the presentation.

4. Conclusions

The assignment gave an idea of how distributed hash tables operate and what one needs to consider regarding problem handling with it. These problems are handling failing nodes and stabilizing the ring. It was challenging, probably one of the toughest tasks I have done for a very long time.

Reference

Coulouris, George F., Dollimore, J., Kindberg, T., Computer scientist. (2012). *Distributed systems: Concepts and design* (5., International ed.). Pearson Education ; Addison-Wesley.