ID2201 Lab 5 Report

Victor Bodell

October 9, 2019

1 Introduction

In Lab 5 I've created a Distributed Hash Table (DHT). In essence a Hash table where the key-value pairs are distributed on different nodes throughout a distributed system. This particular implementation does not really use a hashing technique, since it is only for demonstrational purposes of how distribution relates to storage the implementation in itself is a linked list of nodes, and a simple list of tuples-structure for the key-value pairs.

2 Main problems and solutions

I did not run into any issues regarding the implementation, more than having to carefully consider how the `stabilize`-procedure occurred. This is demonstrated in the following subsection. Other than that I would have enjoyed implementing the fault-tolerant (at least partly) system for replacing crashed nodes, but time unfortunately did not allow this.

2.1 Stabilize!

In order for the ring to stabilize when a new node is added the following procedure takes place for all nodes.

1. Node A: Send out a stabilize-message to self every second, this in turn triggers a `request` from the successor to report it's predecessor.
2. Node B: receive a `request` to report predecessor, report it.
3. Node A: If A's Successors predecessor is nil or B itself, `notify` it to use A as it's successor. If it's A, all is well. If it points to another node C, check whether the other node is a better fit for a predecessor of B than I am. If it is, use C as successor and notify C to use A as predecessor. If it isn't, tell B to use A as a predecessor instead.
4. Node B: If a new predecessor is suggested, check that it is valid. If it is use it, if not ignore it.

3 Evaluation

Evaluating the implementation was simply a matter of using the provided `test` module. See the printout snippet below for some basic tests.

```
2> N = test:start(node2).
<0.64.0>
3> test:start(node2, 10, N).
ok
4> N ! probe.
Probe full circle in 36 micros, nodes:[98,86,3]
probe


10> N ! probe.
probe
Probe full circle in 154 micros, nodes:[98,86,82,78,49,42,32,28,20,15,3]

12> Keys = test:keys(10).
"+'_^`=8F\rJ"
13> test:add(Keys, N).
ok
14> test:check(Keys, N).
10 lookup operation in 0 ms
0 lookups failed, 0 caused a timeout
ok

15> Keys2 = test:keys(100).
[14,43,59,52,54,10,39,82,46,85,5,41,95,64,44,74,61,38,76,94,
 10,50,71,34,43,56,3,62,14|...]
16> test:add(Keys2, N).
ok
17> test:check(Keys2, N).
100 lookup operation in 1 ms
0 lookups failed, 0 caused a timeout
ok
```

Below follows some comments on what happens in the tests. Note that for the `key:generate\0` function values between 0-100 were used rather than the recommended 1.000.000.000 for readability.

- `3>` Ten nodes are created in the ring and attached to the original node `N`.
- `4>` Initial probing is too quick and the ring hasn't had time to stabilize.

- `10>` After a few seconds the ring has stabilized, and probing returns a (reverse) sorted list of how the message has traveled, yielding a sorted ring. Note that all 11 nodes are part of the ring. If one would have gotten a number the same as another this would have resulted in that node not becoming part of the ring due to the implementation of `stabilize`. This is because Id-values must be unique.
- `14>` 10 keys have been randomly created, added to the ring, and lookedup. The tests show that all messages were found. The only thing the tests state is that it asked for the keys and got them back from someone in the ring. It could be that the first node handles all the keys and that the keys are therefor not propagated within the ring. For this one should conduct a test that states which node returned the lookup.
- `17>` Same as `14>` for 100 keys.

4 Conclusions

I found this lab fun to implement and useful, albeit with a lot of skeleton code and help, making it less challenging and more about trying to understand copied code – although this is probably closer to reality than the other scenario of implementing the thing from scratch yourself.

I learned some structures in keeping a ring organized in a distributed system by a single algorithm. There was some thought process necessary in order to understand the structure of the DHT.