HW5 - Chordy

Reethika Ambatipudi

11th October 2022

1 Introduction

This assignment is more related to the chord and Distributed hash table.We start a group of nodes. All the nodes are given IDs. The buckets of the DHT are shared among all the nodes.The logic behind distribution of the buckets is

```
PredecessorID < Key <= NodeId.
```

Client can ask any of the nodes for a particular key and nodes have to communicate among themselves through message passing, search for the asked key and they return the key,value to the client. There could be peer churn and to handle this, we implement stabilization periodically where all the nodes ask their successor who their predecessor is. If stabilization is not done periodically, new connections will not be updated and messages will be lost. The whole process of searching for a key,value pair will not progress.

2 Main Problems and Solutions

2.1 node1 - Basic ring structure

In this module we start a node, other nodes are added to this node and the whole ring is formed. Initially, we set our predecessor to nil and then be run the stabilization process. Then we connect the nodes. If we are the only node in the ring, we become our own successor. A probe message can be sent to check if the ring is really connected.

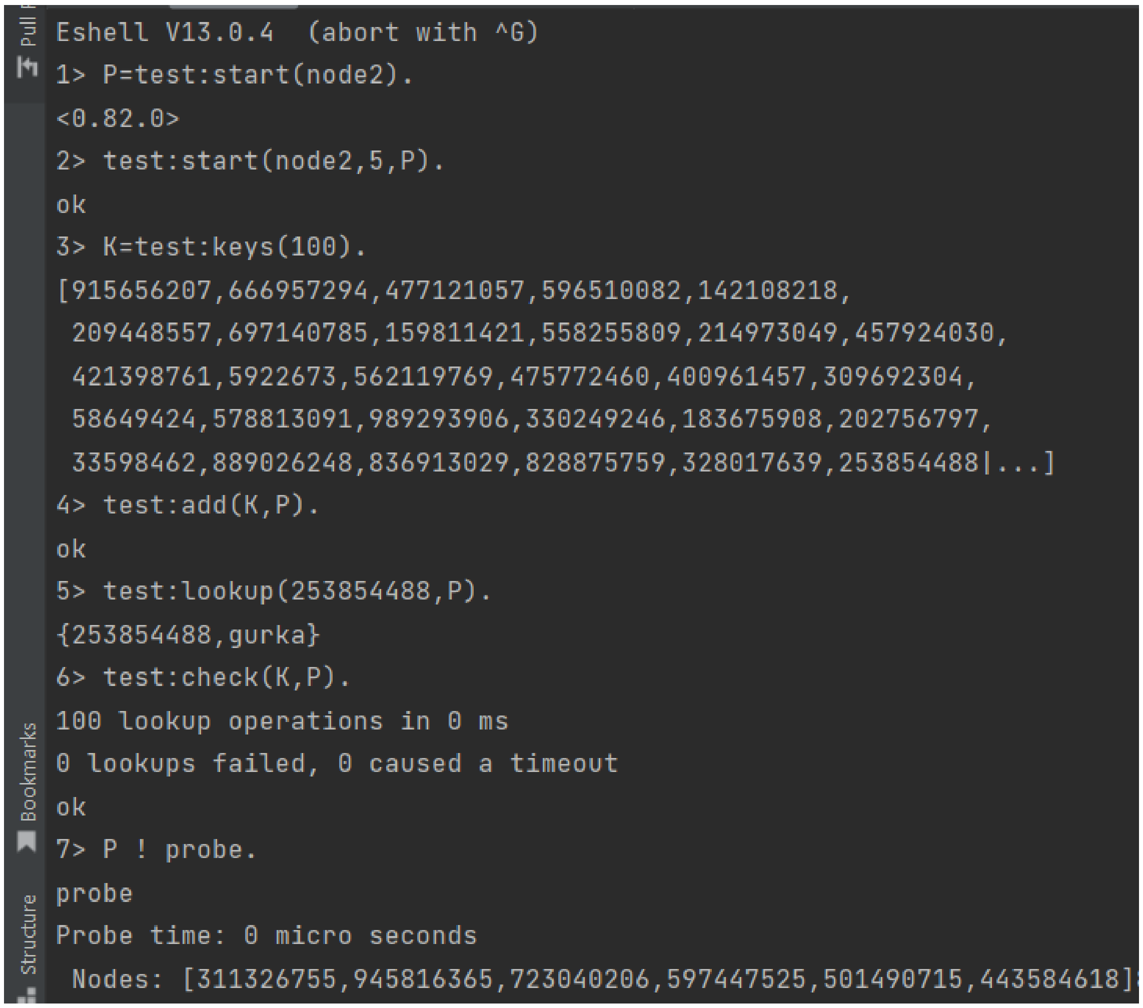
2.2 node2 - Storage implementation

In this module we implement the storage, generate some keys with the help of key module, assign values and then add them to the storage. The buckets are distributed among the nodes. We can search for a particular key with the help of lookup function and key,value is returned. We can also check the throughput with the help of check function in the test module.

2.3 node3 - Failure detection

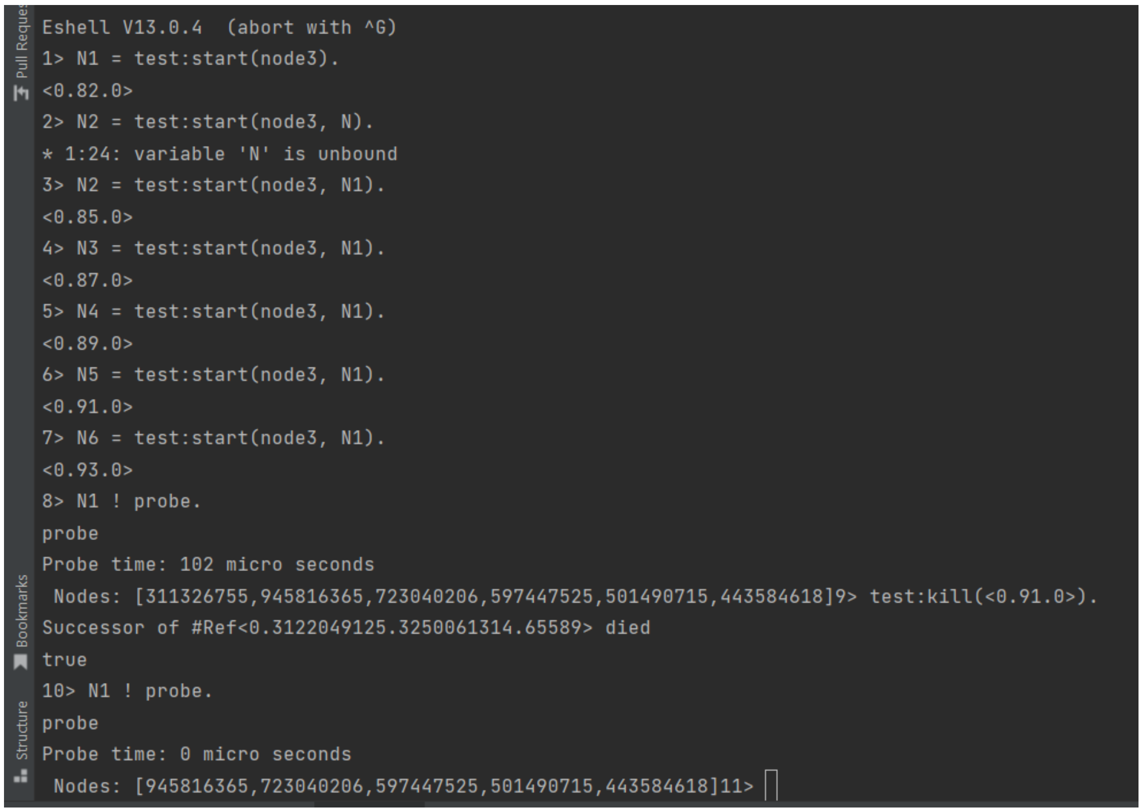
Here we have a monitor to check if all the nodes are alive. If the monitor detects failure of a node, it immediately does stabilization and updates new connection of the ring.

3 Evaluation



```
Eshell V13.0.4  (abort with ^G)
1> P=test:start(node2).
<0.82.0>
2> test:start(node2,5,P).
ok
3> K=test:keys(100).
[915656207,666957294,477121057,596510082,142108218,
 209448557,697140785,159811421,558255809,214973049,457924030,
 421398761,5922673,562119769,475772460,400961457,309692304,
 58649424,578813091,989293906,330249246,183675908,202756797,
 33598462,889026248,836913029,828875759,328017639,253854488|...]
4> test:add(K,P).
ok
5> test:lookup(253854488,P).
{253854488,gurka}
6> test:check(K,P).
100 lookup operations in 0 ms
0 lookups failed, 0 caused a timeout
ok
7> P ! probe.
probe
Probe time: 0 micro seconds
 Nodes: [311326755,945816365,723040206,597447525,501490715,443584618]
```

Figure 1: Network of 6 nodes



```
Eshell V13.0.4  (abort with ^G)
1> N1 = test:start(node3).
<0.82.0>
2> N2 = test:start(node3, N).
* 1:24: variable 'N' is unbound
3> N2 = test:start(node3, N1).
<0.85.0>
4> N3 = test:start(node3, N1).
<0.87.0>
5> N4 = test:start(node3, N1).
<0.89.0>
6> N5 = test:start(node3, N1).
<0.91.0>
7> N6 = test:start(node3, N1).
<0.93.0>
8> N1 ! probe.
probe
Probe time: 102 micro seconds
 Nodes: [311326755,945816365,723040206,597447525,501490715,443584618]9> test:kill(<0.91.0>).
Successor of #Ref<0.3122049125.3250061314.65589> died
true
10> N1 ! probe.
probe
Probe time: 0 micro seconds
 Nodes: [945816365,723040206,597447525,501490715,443584618]11>
```

Figure 2: monitor

4 Conclusion

There are still some problems to be addressed. If a node fails, or abruptly leaves the ring, there also might be loss of the data. In order to avoid this, every node should have a duplicate of the data of it's predecessor. This could help in retrieving the data when a node dies. Overall, this assignment was very interesting and useful in understanding the whole process of chord and DHT.