Project - week 2

REST interface and MapReduce

February 23, 2017

Students:
Mees Kalf
10462074

Coordinator:
dhr. L. Torenvliet

Course:
Modern Databases

Catalog number:
5062MODA6Y

1 Introduction

1.1 Describe the structure of a Chord overlay network. What kind of information does each node contain and in what way are they linked with each other?

A Chord overlay network is a distributed protocol/algorithm for storing a distributed hash table in multiple computers. The structure is based on hash functions evenly dividing keys over each node, computer, which provides a degree of natural load balance. Each node has a successor and a predecessor in a ring, when refer to ring I mean this indentifier ring, formation, the last node is the predecessor of the first node and the first node is the successor of the last node.

Each node only has to be aware of it's successor and contains a so-called finger table. This finger-table consists of a table with the hash for other nodes in the network. Each next node in the table being n^2 apart, considering the ring formation, from the previous node. A node obtains the finger-table through it's successor.

1.2 The Chord overlay network is considered scalable. Explain why this is true.

In the previous section I mentioned that each node keeps track of a finger-table containing different nodes on the ring. Since each node in the finger-table is a power of two apart the finger table consists of $O(logn)$, considering a ring of size n. So while the ring grows exponentially the finger-tables grow linear which makes the Chord algorithm suitable for scaling to large numbers.

1.3 Explain what happens when a node joins an existing Chord overlay network.

When a new node n connects with node m to join the ring n asks m to look up it's predecessor and finger table. To reduce the request time m simple constructs the finger-table by looping over it's own finger table asking for their predecessor, since n will be the new predecessor of m the finger tables are almost identical only shifting one node.

After n being initialized all other finger-tables need to updated with n this is done by looping over every entry in it's finger table, checking their predecessor and updating their finger table.

1.4 Describe the purpose of each of the following methods used in the aforementioned template code.

stabilise	This functions keeps track of the pointers of it's successor and checking if it's successors finger table is correct and stabilises when changes are noticed.
notify	Notifies it's successor of it's existence.
fix_fingers	When a new node enters the ring this function fixes the finger-tables which need to be fixed.

1.5 Consider a stable Chord network with a ring size of 32 that contains the following set of node IDs 4,7,17 18,19. Show the predecessor and the fingers for each of the nodes.

Finger	Next finger	Successor
5	6	7
6	8	7
8	12	17
12	20	17
20	4	4

Table 1: Finger-table node 4 with predecessor 19

Finger	Next finger	Successor
8	9	17
9	11	17
11	15	17
15	23	17
23	7	4

Table 2: Finger-table node 7 with predecessor 4

Finger	Next finger	Successor
18	19	18
19	21	19
21	15	4
25	23	4
1	17	4

Table 3: Finger-table node 17 with predecessor 7

Finger	Next finger	Successor
19	20	19
20	22	4
22	26	4
26	2	4
2	18	4

Table 4: Finger-table node 18 with predecessor 17

Finger	Next finger	Successor
20	21	4
21	23	4
23	27	4
27	3	4
3	19	4

Table 5: Finger-table node 19 with predecessor 18

1.6 Consider a stable Chord network with a ring size of 32 that contains the following set of node IDs 6,7,22. Show the predecessor and the fingers for each of the nodes.

z

Finger	Next finger	Successor
7	8	7
8	10	22
10	14	22
14	23	22
22	6	22

Table 6: Finger-table node 6 with predecessor 22

Finger	Next finger	Successor
8	9	22
9	11	22
11	15	22
15	23	22
23	7	6

Table 7: Finger-table node 7 with predecessor 6

Finger	Next finger	Successor
23	24	6
24	26	6
26	30	6
30	6	6
6	22	6

Table 8: Finger-table node 22 with predecessor 7

1.7 Illustrate what happens when the node with node ID 6 leaves the network due to a failure by showing the predecessor and the fingers for each of the nodes

Finger	Next finger	Successor
23	24	22
24	26	7
26	30	7
30	6	7
6	22	7

Table 9: Finger-table node 22 with predecessor 7

Finger	Next finger	Successor
8	9	22
9	11	22
11	15	22
15	23	22
23	7	7

Table 10: Finger-table node 7 with predecessor 22

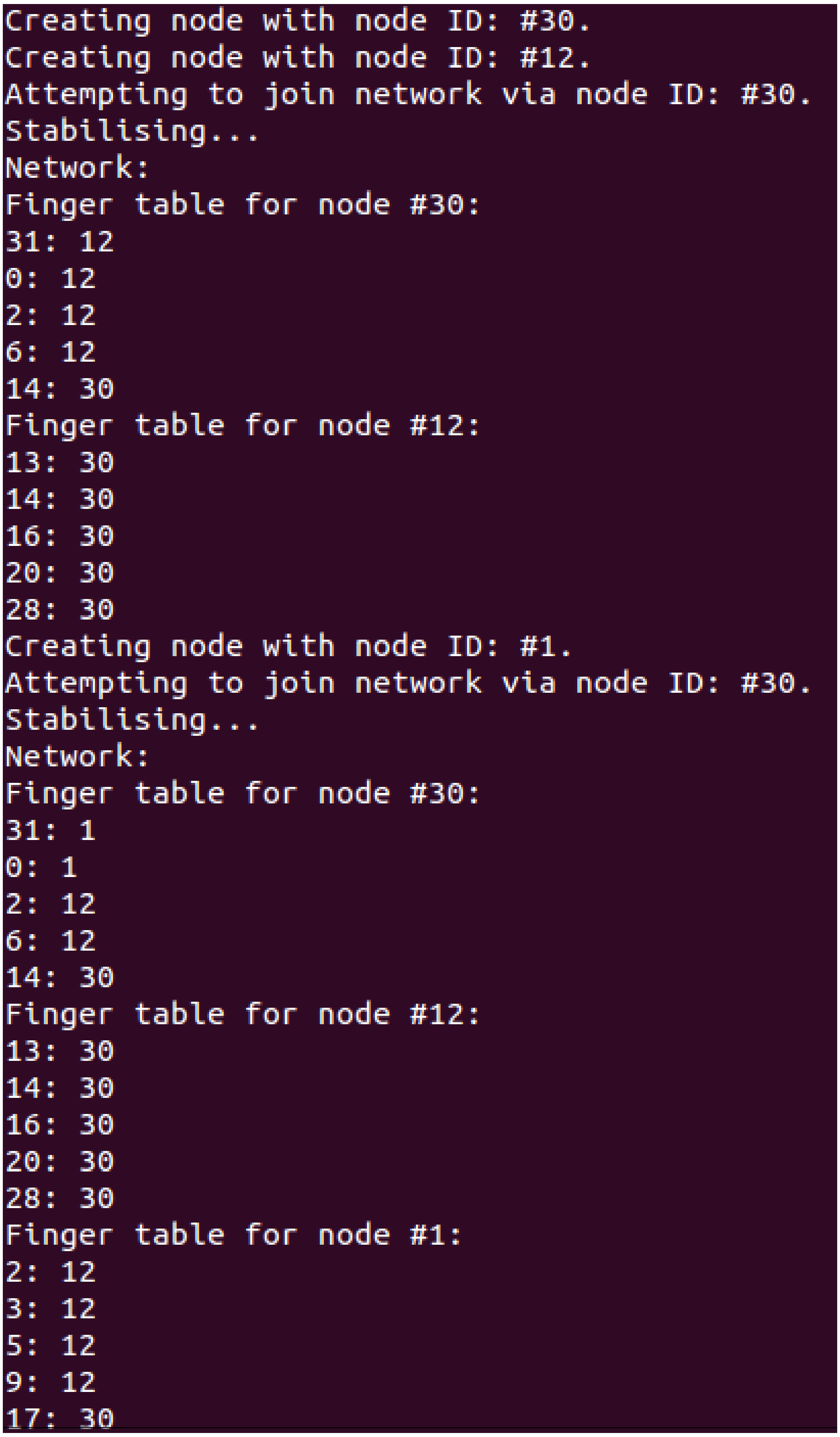
2 Implementing the Chord algorithm

2.1 Struggles

This is actually the first exercise of the YAMR project I found quite doable as an individual. The paper [1] was really good readable and most functions could easily be written in python(right from the paper). Only the stabilize function needed some more attention for checking if nodes were still alive. Still it took quite some time to really understand the Chord algorithm so well to actually get everything to work. I would've liked some documentation at the template code, the first two hours I was quite hopeless from where to start and what the framework actually did, although I admit it's actually quite good readable. I wasn't really the coding i struggled with but more adding 1 for looking at the right node_id on which my brain practically crashed. It might have been better if we would've first have to make the implementation and afterwards do the exercises. Since I got a way better view of the Chord algorithm after the implementation than before.

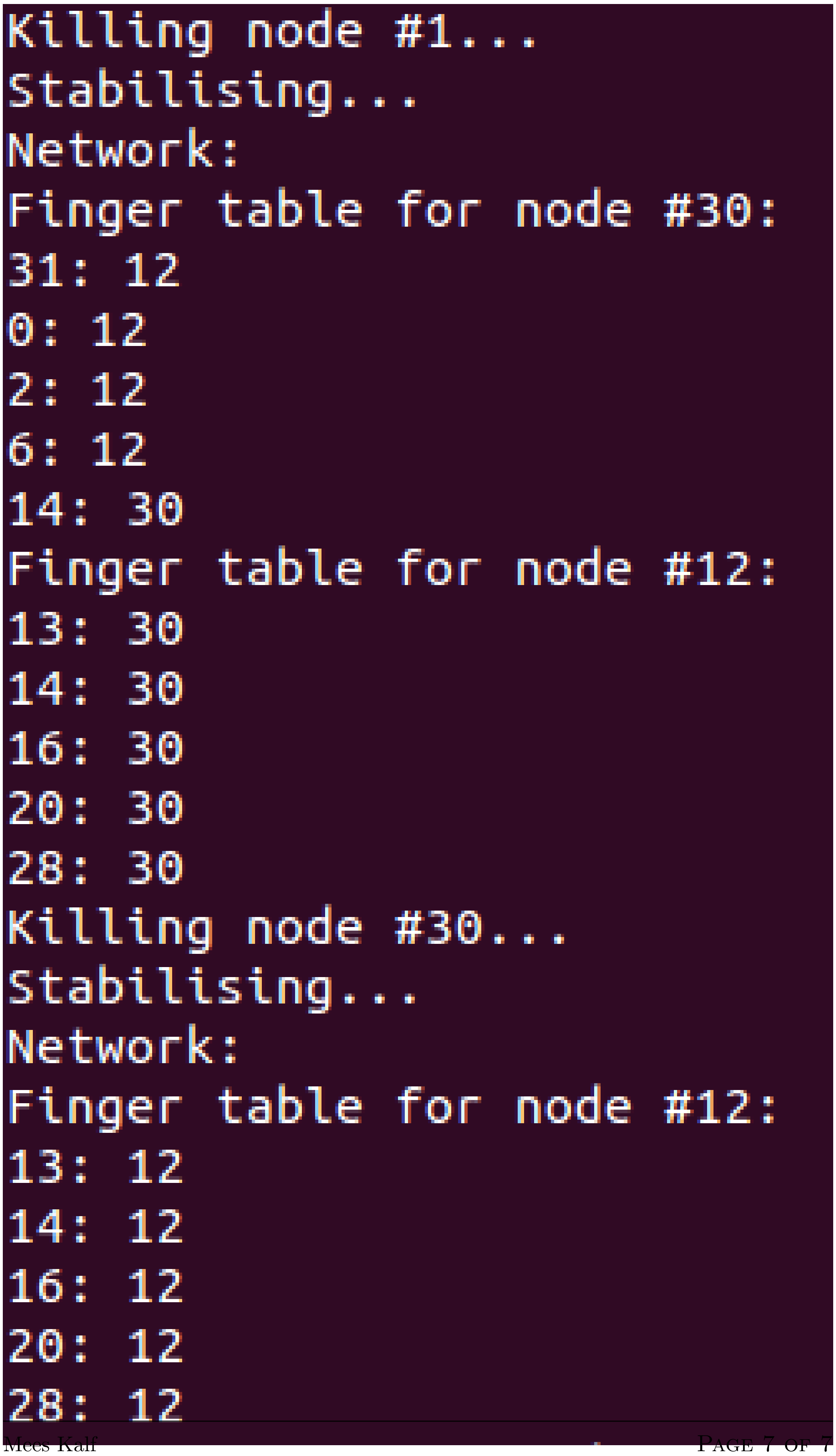
2.2 Results

[1] Ion Stoica, Robert Morris, David Karger, M Frans Kaashoek, and Hari Balakrishnan. Chord: A scalable peer-to-peer lookup service for internet applications. ACM SIGCOMM Computer Communication Review , 31(4):149160, 2001.



```
Creating node with node ID: #30.
Creating node with node ID: #12.
Attempting to join network via node ID: #30.
Stabilising...
Network:
Finger table for node #30:
31: 12
0: 12
2: 12
6: 12
14: 30
Finger table for node #12:
13: 30
14: 30
16: 30
20: 30
28: 30
Creating node with node ID: #1.
Attempting to join network via node ID: #30.
Stabilising...
Network:
Finger table for node #30:
31: 1
0: 1
2: 12
6: 12
14: 30
Finger table for node #12:
13: 30
14: 30
16: 30
20: 30
28: 30
Finger table for node #1:
2: 12
3: 12
5: 12
9: 12
17: 30
```

Figure 1: Showing the creation of 3 nodes with the Chord algorithm in a 32 sized ring



```
Killing node #1...
Stabilising...
Network:
Finger table for node #30:
31: 12
0: 12
2: 12
6: 12
14: 30
Finger table for node #12:
13: 30
14: 30
16: 30
20: 30
28: 30
Killing node #30...
Stabilising...
Network:
Finger table for node #12:
13: 12
14: 12
16: 12
20: 12
28: 12
```

Figure 2: Killing 2 of the 3 nodes with the Chord algorithm in a 32 sized ring