Report 5: Chordy - a distributed hash table

Yining Hou

October 10, 2023

1 Introduction

In this report, I present a distributed hash table in 4 steps.

In this seminar, the main topic is about how to maintain a ring structure in distributed systems. We will be able to add nodes in the ring and add and search for values. I also add failure detection to the system.

2 Building a ring

The first implementation will only maintain a ring structure; we will be able to add nodes in the ring but not add any elements to the store.

2.1 Start one node

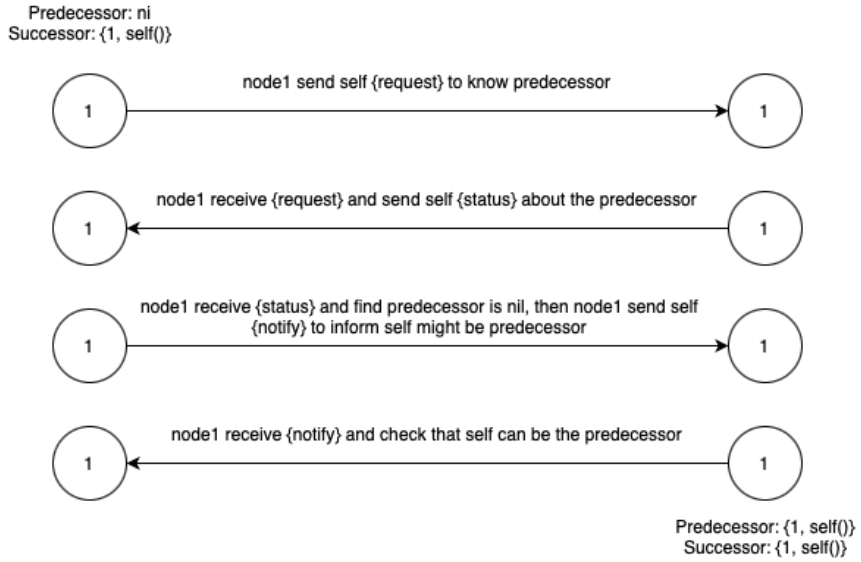

Figure 1: one node in the ring

2.2 A node join

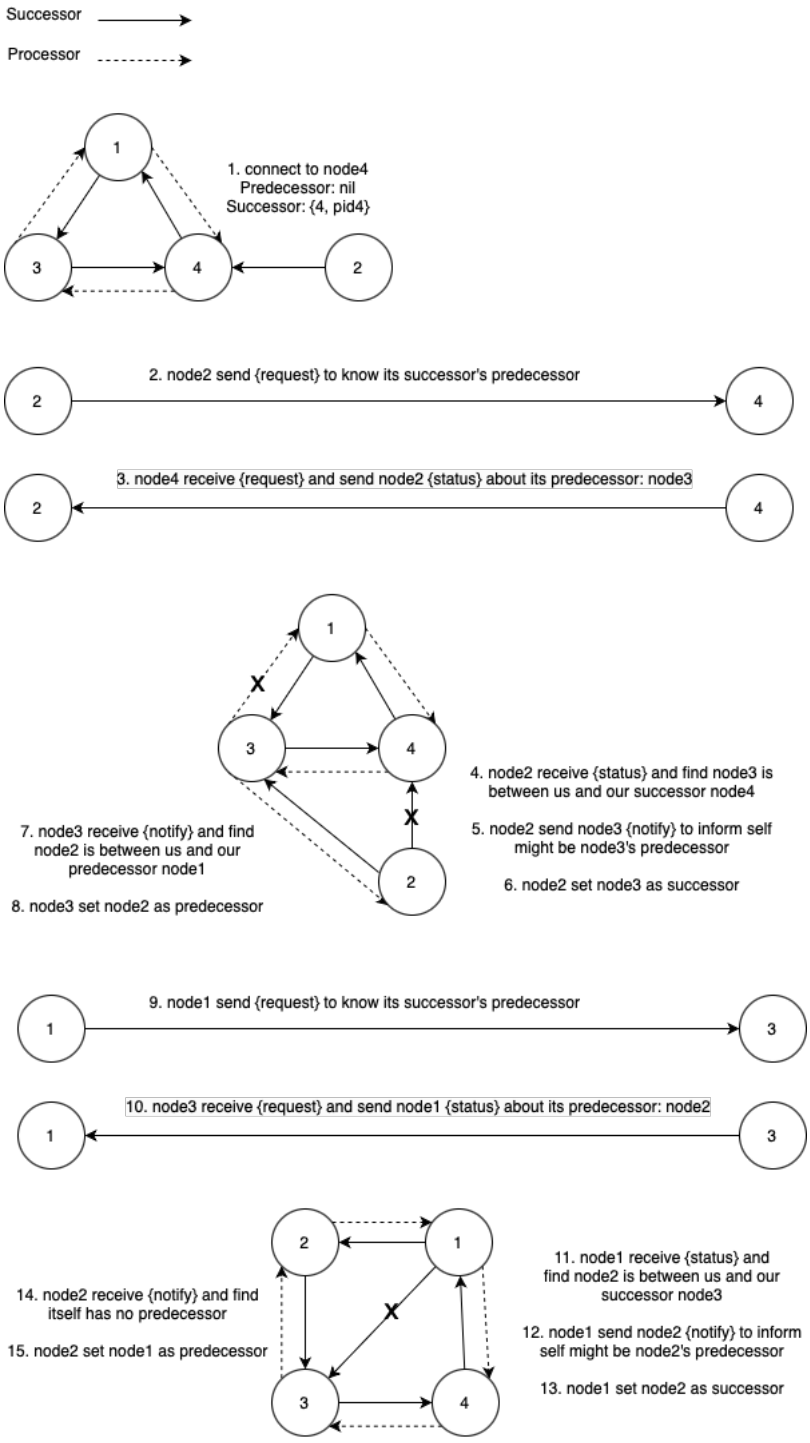

Figure 2: a new node join

2.3 Evaluation

When the ring is built, introduce a probe message to check if the ring is actually connected.

```
(chordy@130.229.160.184)1> register(node, test:start(node1)).
true
(chordy@130.229.160.184)2> node ! probe.
probe
Probe: 1277µs
Nodes: [872344517,430003878,554715116]

(chordy2@130.229.160.184)1> test:start(node1, 2,
{node, 'chordy@130.229.160.184'}).
ok
430003878 forward probe
554715116 forward probe
```

3 Adding a Store

Introduce a store where key-value pairs can be added. Adding and searching for values will only introduce a few new messages and one parameter representing the store.

A new node should take over already added elements. When a node receives a notify message and accepts a new predecessor, it should split its Store based on the new predecessor's key. The node keeps a store contains (NewPredecessorKey, NodeId], and (OldPredecessorKey, NewPredecessorKey] will be handed over to our new predecessor.

3.1 Performance

As a first test, we can have only one node in the ring and let the four test machines add 1000 elements to the ring and then do a lookup of the elements.

```
10> test2:performance1(1000).
Test Machine1:<0.138.0> add 1000 elements
Test Machine2:<0.139.0> add 1000 elements
Test Machine3:<0.140.0> add 1000 elements
Test Machine4:<0.141.0> add 1000 elements
<0.136.0>
Test Machine1: finish in 142 ms
Test Machine2: finish in 142 ms
Test Machine3: finish in 143 ms
Test Machine4: finish in 143 ms
```

As the second test, one machine handles 4000 elements.

```
11> test2:performance2(4000).
Test Machine1:<0.145.0> add 4000 elements
<0.143.0>
Test Machine1: finish in 142 ms
```

The time spent is almost the same. I think the limiting factor is the time cost of a node to handle each request.

4 Bonus1 - Handling Failures

If our successor dies, we have to contact the next in line. So we keep track of one more successor - Next.

The Next node will not change unless our successor send a request message to informs us about a change.

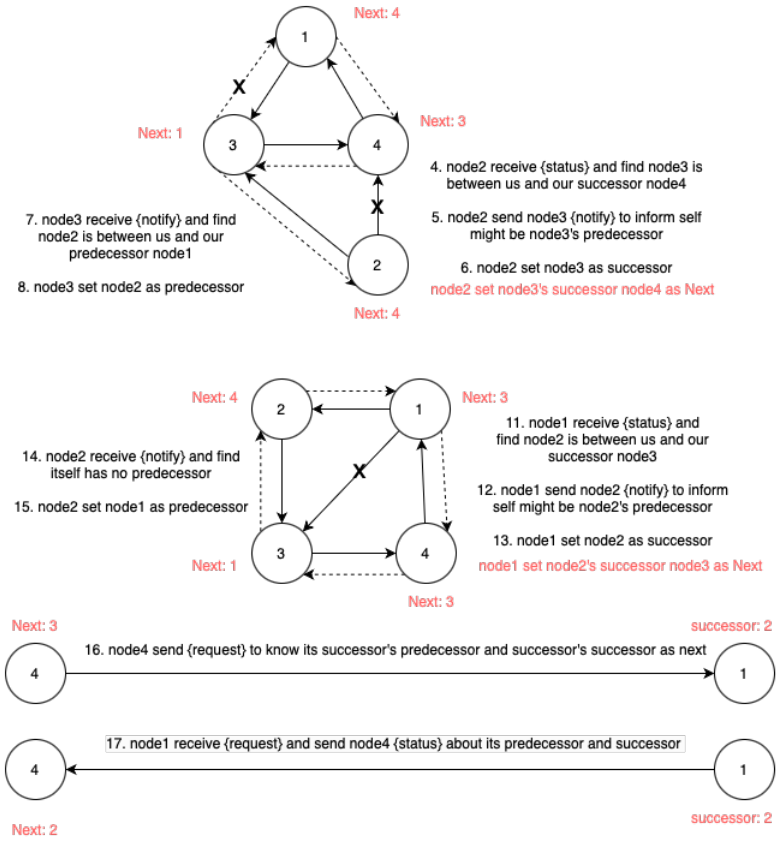

Figure 3: update next node in the ring

Send `kill` message to simulate failure. We need to keep track of both our successor and predecessor. When a node dies, we need to de-monitor from it and monitor our new successor and predecessor.

4 places to create a monitor:

- In `init` when we connect to a new successor;
- In `stabilize` when we change our successor;
- In `notify` when we adopt a predecessor;
- In `notify` when we change our predecessor.

2 places to demonitor:

- In `stabilize` when we change our successor;
- In `notify` when we change our predecessor.

When a node obtains a `DOWN` message, if its predecessor dies, it will set its predecessor to `nil` since someone will sooner or later present themselves as possible predecessor. If its successor dies, adopt its next node as successor.

```
(chordy@130.229.160.184)107> <0.3626.0> ! node_status.
735811334 Pre: {602662153,#Ref<0.805936275.1258291201.155483>,<0.3634.0>},
Suc: {867106540,#Ref<0.805936275.1258029058.110505>,<0.3631.0>},
Next: {484355151,<0.3632.0>}, Store: []
(chordy@130.229.160.184)111> <0.3632.0> ! {kill, 867106540}.
{kill,867106540}
(chordy@130.229.160.184)112> <0.3632.0> ! node_status.
484355151 Pre: {735811334,#Ref<0.805936275.1258291201.163681>,<0.3626.0>},
Suc: {602662153,#Ref<0.805936275.1258029058.110435>,<0.3634.0>},
Next: {735811334,<0.3626.0>}, Store: []
 (chordy@130.229.160.184)113>  <0.3632.0> ! probe.
602662153 forward probe
735811334 forward probe
Probe: 470µs
Nodes: [484355151,602662153,735811334]
```

5 Bonus2 - Replication

If a node dies, we will lose information. To solve this problem, we will have to replicate the values in the store.

When we add a key-value element to our own store we also forward it to our successor as a replicate, Key, Value message. Each node will thus have a second store called the Replica where it can keep a duplicate of its predecessor's store.

5.1 New node join and replication

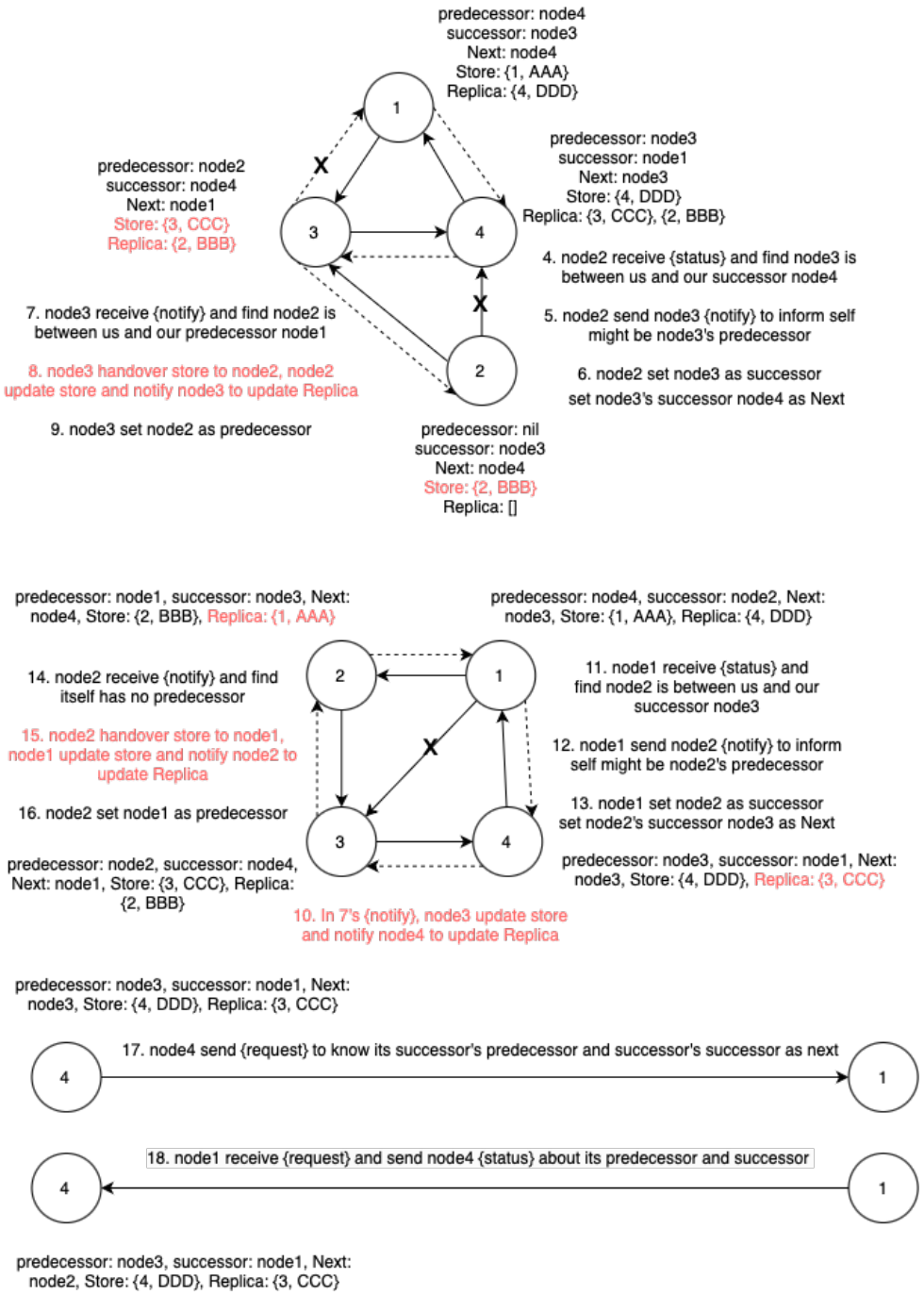

Figure 4: a new node join

5.2 A node down

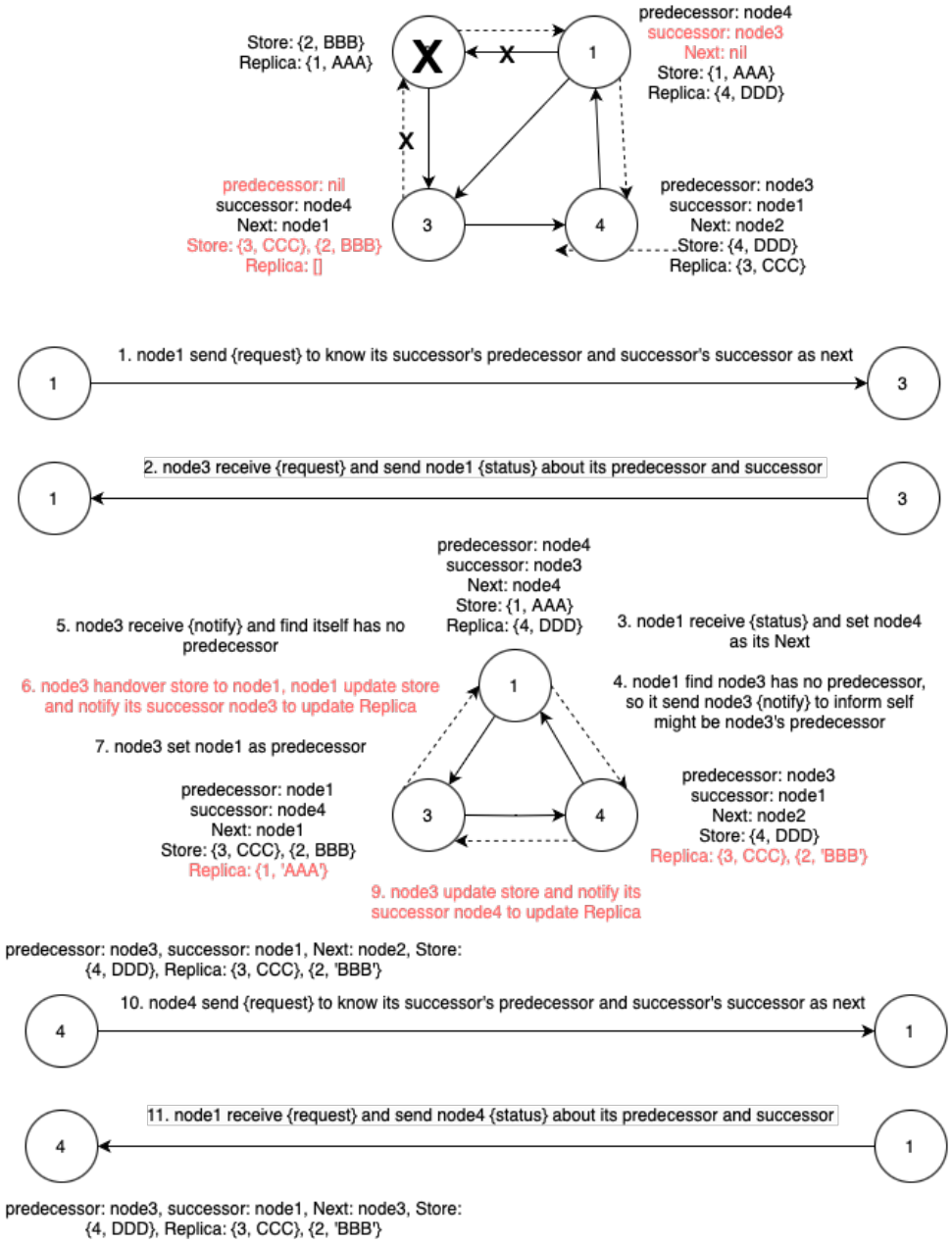

Figure 5: a node is down

6 Conclusions

Chordy works well and certain methods have been added to handle failures in nodes.

It's already quite complicated to figure out how nodes are stabilized into a ring. However, when I step into the bonus part, things become even more messy. It takes a lot of patience to correctly track the Next node and Replication of the store. I need to pay attention to every details and go through the whole process again and again. It's a large amount of work to make a 'perfect' ring.