Report 5: chordy

Xiaoying Sun

October 7, 2020

1 Introduction

The main tasks in this assignment are as follows:

1) Add nodes in the ring

2) Create distributed stores where key-values pairs can be added

3) Add elements to the store

4) Probe if the distributed system works

5) Add failure detection

6) Introduce a simple replication scheme

7) Introduce a routing table

2 Main problems and solutions

2.1 Adding nodes

The interval of generated process ID is 2, because each node has a monitor process.

What does it means that $Xkey$ is equal to $Skey$, will it happen? Does that matter?

Our successor uses a predecessor which points to itself. Let's notify him, that he can point to us.

```
2> A = node1:start(0).
<0.87.0>
0: NOTIFY by {0,<0.87.0>}
3> C = node1:start(2, A).
<0.89.0>
2: 0, point to me!
0: NOTIFY by {2,<0.89.0>}
0: 2, point to me!
2: NOTIFY by {0,<0.87.0>}
```

Figure 1: A adding C

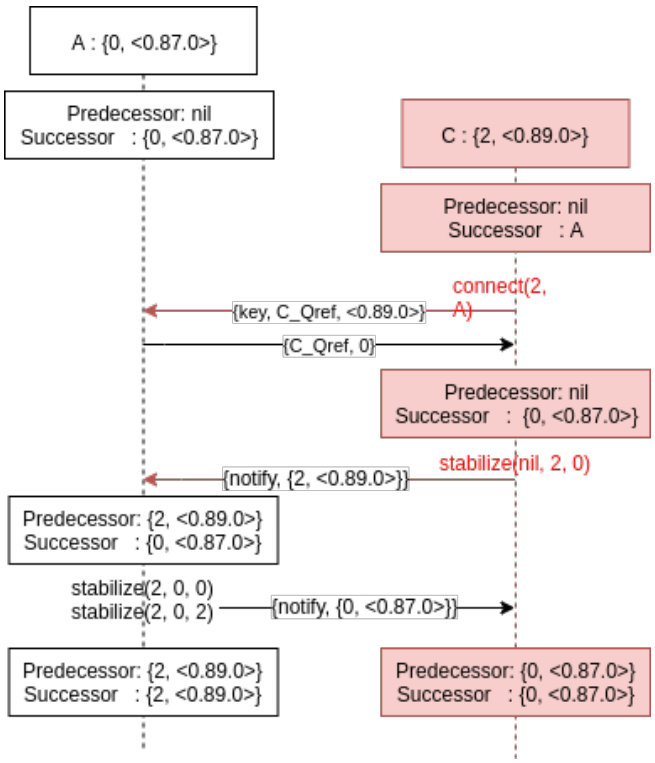

Figure 2: Messages between A and C

What if the key of the predecessor of our successor($Xkey$) is between us and our successor?

We should adopt this node as our successor and run stabilization again.

```
2> A = node1:start(0).
<0.87.0>
3> C = node1:start(2, A).
<0.89.0>
2: 0, point to me!
0: 2, point to me!
4> B = node1:start(1, A).
<0.91.0>
1: 2, point to me!
0: 1, point to me!
5> D = node1:start(3, A).
<0.93.0>
2: 3, point to me!
```

Figure 3: Ring AC adding B

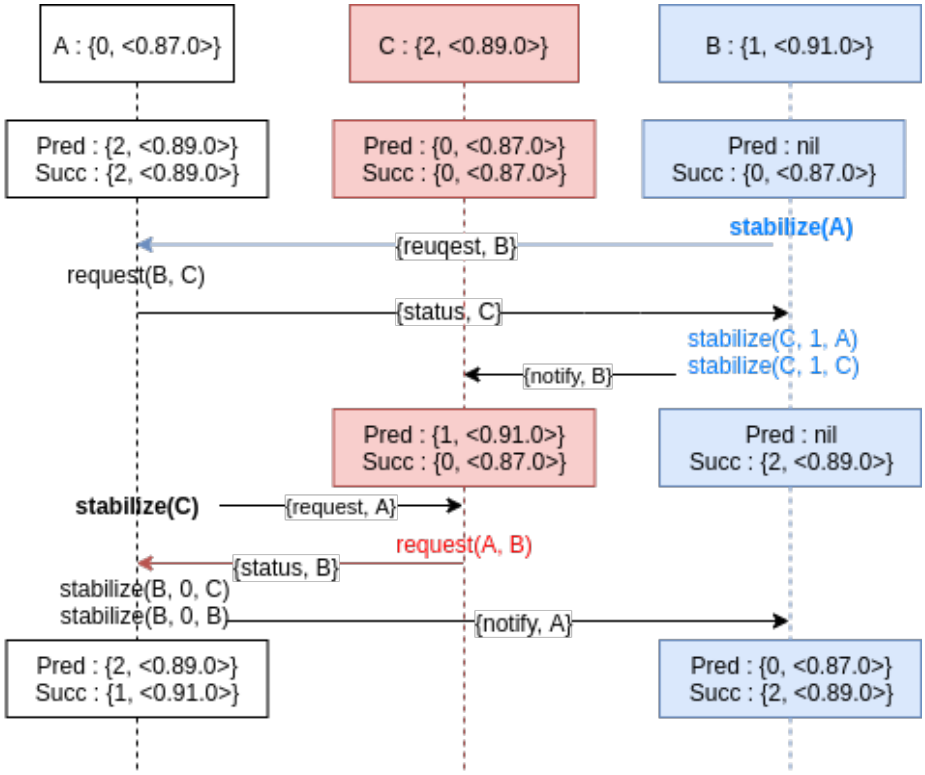

Figure 4: Messages between A, C and B

2.2 Adding a store

Which part should be kept and which part should be handed over to the new predecessor?

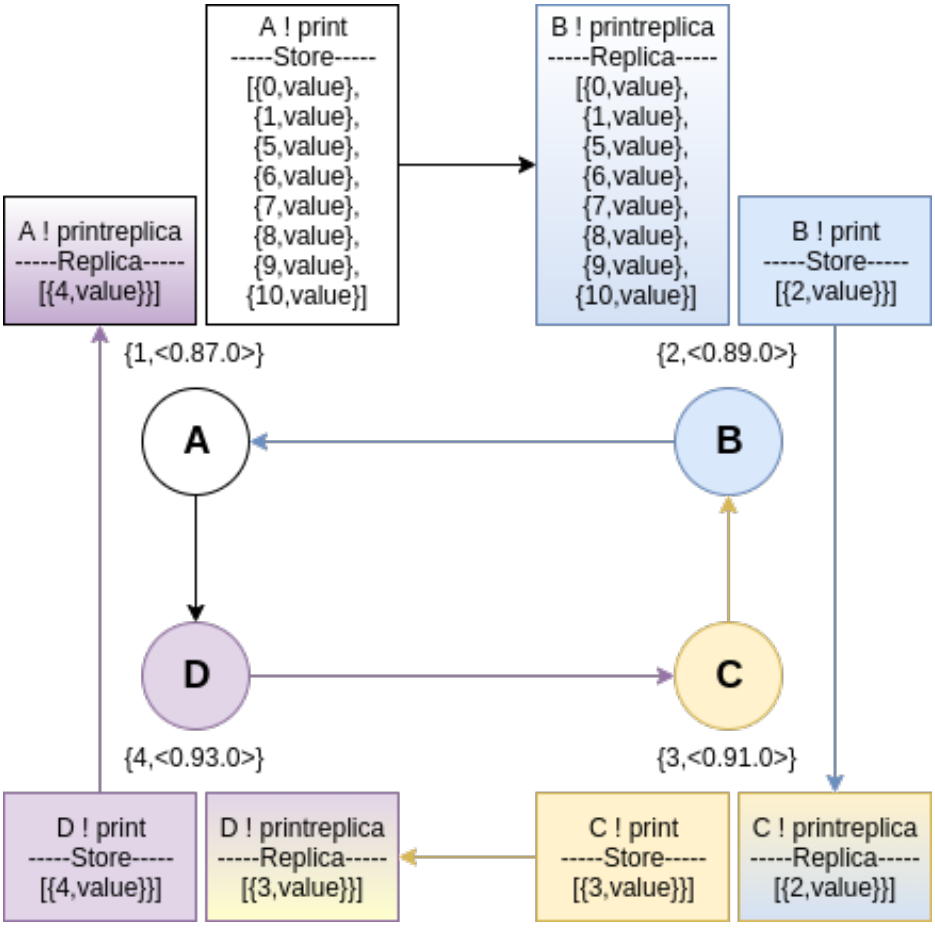

Figure 5: Stores and replications

2.3 Handling failures

Failures	Solutions
predecessor stops	set our predecessor to nil
successor stops	adopt next node
temporarily unavailable	wait for TIMEOUT

Table 1: Solutions of failures

If a node dies, it's successor is of course responsible for the store hold by the node. We have to remove the old monitor and create a new one, pointing to the new successor.

It is also important that we give away our current store as a replica for our new successor.

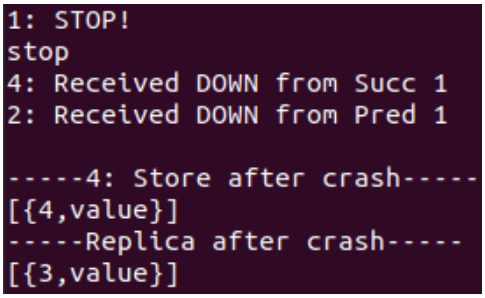

Figure 6: Successor stops

If our predecessor dies, we have to take care of its keys, means, we merge our store with the replica! Since we have new keys in our store, we also have to inform our successor to add these keys to his replica.

1. Merge our store with replica!
2. Inform our successor to add these keys to his replica.
3. Find new predecessor.
4. Split *Store* at $Nkey$ and hand it over to new predecessor.
5. Split *Replica* at $Nkey$ and hand it over to new predecessor.

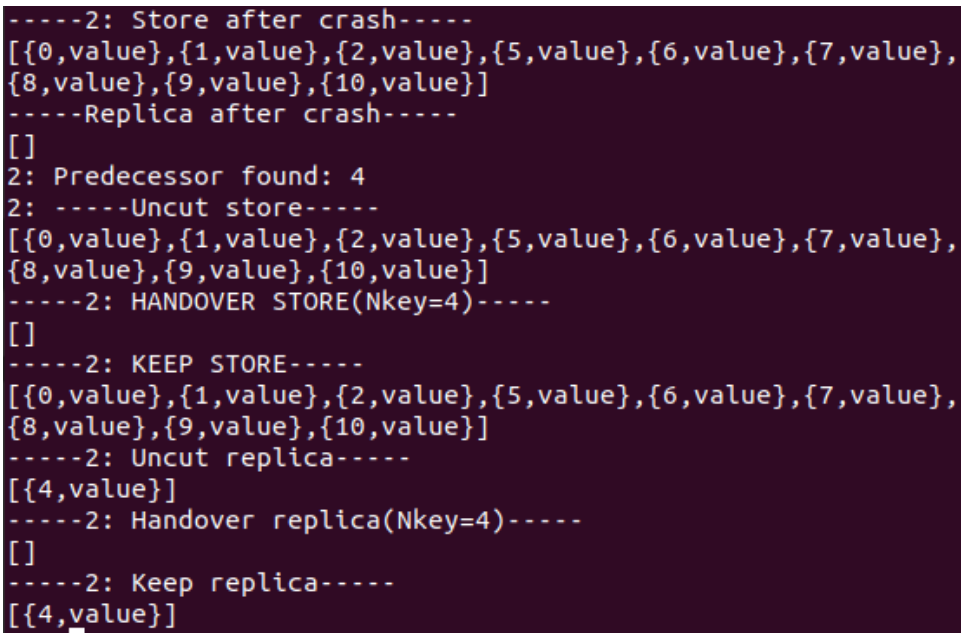

Figure 7: predecessor stops

3 Evaluation

Does it take longer for one machine to handle 4000 elements rather than four machines do 1000 elements each?

(N_node, N_write)	Time			Aver
(1, 1000)	116.22	105.529	103.971	108.573
(2, 500)	70.6085	70.443	70.185	70.412
(5, 200)	49.7446	47.7054	49.5094	48.986
(10, 100)	47.2228	45.1016	47.2788	46.5344
(20, 50)	42.8789	42.787	41.95	42.5386

Table 2: Solutions of failures

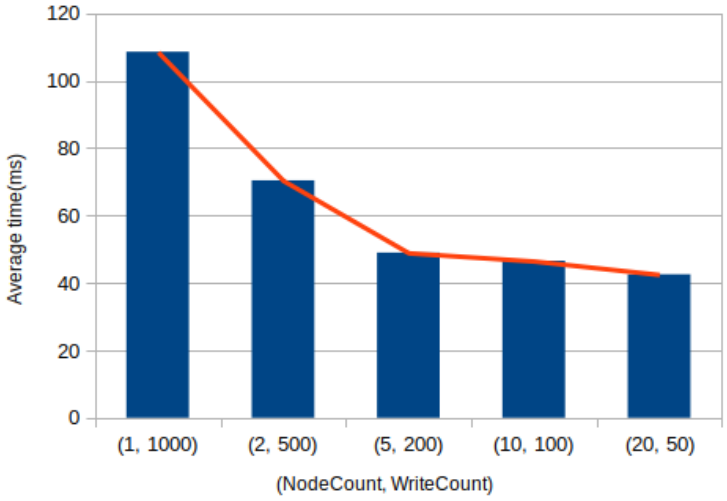

Figure 8: predecessor stops

One of the tests made is illustrated in Figure 8. A number of 1000 successive read and write commands had been issued to a different number of concurrently running nodes. **The more nodes, the less time is taken to get a response.** It is necessary to mention, that the total number of requests was distributed amongst all nodes.

4 Conclusions

From the above, we successfully implemented a distributed hash table following the Chord scheme, which not only periodic stabilizes the ring, but could also tolerate various kinds of complicated errors.

Most of my time spent on understanding how to bring a new node into the right order and make its data store accessible to clients. The key is the function

$$stabilize(Pred, Id, Successor)$$

, which decides the predecessor and successor of a node.

5 References

- http://erlang.org/doc/man/erlang.html
- https://zhuanlan.zhihu.com/p/53711866
- Stoica, I., Morris, R., Karger, D., Kaashoek, M.F. and Balakrishnan, H., 2001. Chord: A scalable peer-to-peer lookup service for internet applications.ACM SIGCOMM Computer Communication Review,31(4), pp.149-160.