Group 8

Capella

Distributed Datastore

Implementation details

Harshit Sikchi, Tejus Gupta, Shikha Panwar, Siddhant Meshram

Mentor : Gulab Arora

17th March, 2019

User Features

1. Sequential Consistency.
2. Read <key, value> pair.
3. Write <key, value> pair.

System specifications

1. No single point of failure.
2. All nodes know IP-address, port-no of all other nodes initially.

System Assumptions

1. A node can crash, but will not come back after crash.
2. We assume that after 1 crash, no crash will happen till the recovery completes. This is reasonable assumption, since our machines rarely crash.
3. The system will work as long as have R live nodes i.e., it is resistant to N-R crash failures where N, R are no. of initial nodes and Replication factor (3 in our case).

System Design Specifications

System Model:

Every key-value pair is associated with a version number that gives the number of times a file has been updated.

Version numbers are stored on stable storage. Every successful write updates version number.

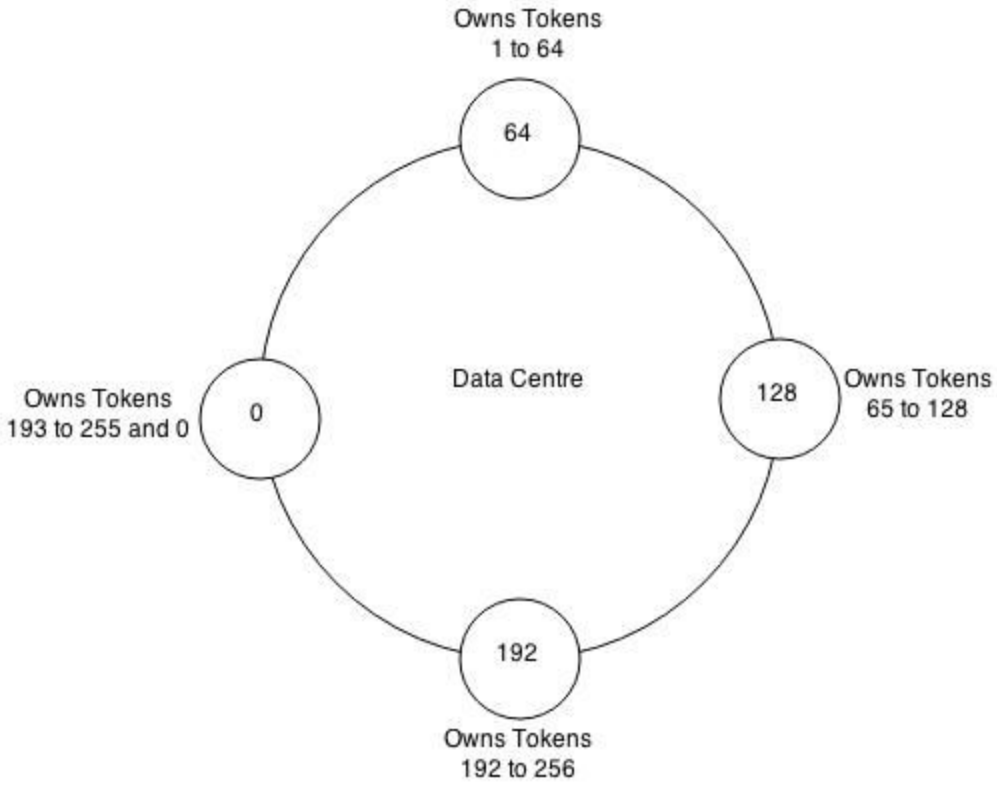
Basic Idea:

A read or write operation permitted if a certain number of votes, called read quorum or write quorum, are collected by the requesting process.

Consistency Model :

- We guarantee sequential consistency.
- We use quorum-based voting algorithm Q_r = 2 and Q_w = 2 to guarantee that sequential consistency is maintained even with single node failure.

Replication Model :



- Data is replicated 3 times(Replication factor). The node with the token hash(k) stores the data with key k, say node i. The data is also replicated at node i+1, and i+2. i is said to own the token hash(k). As we will see, the ownership of token will change when node crash.
- We use a quorum-based voting algorithm for providing sequential consistency. We set Q_r = 2 and Q_w = 2 to guarantee that sequential consistency is maintained even with node failure.

Interaction with Client

Client can read or write key-value pair.

Client requests any node in the system, which acts as co-ordinator. In future work we can do this load-balancing using DNS

Data Structures

Each node maintains the following data structures to keep track of the current ring (structure):

1. Ownership: Each node maintains which all values of hash(key) is it the owner of. Initially: ownership = [its own id]. If a node k fails, the next ownership of next live node extends to include the one of k.
2. Failed_nodes: Each node maintains what all the nodes have failed. This is required for reading and writing from appropriate nodes even after failure.
3. Roll Back database : Value stores previously for roll back purposes.

Algorithm:

Read Algorithm :

- User sends request to coordinator to read key k.
- Coordinator sends coordinator-read message to all the R replicas where key k is stored. These replicas send their value and version back to the coordinator.
- Coordinator waits to receive reply from altleast Qr replicas and then relies to user the value corresponding the latest version.

Write Algorithm (similar to 3 phase commit)

- User sends request to coordinator to write key (k, v).
- Coordinator sends coordinator-write message to all the R replicas where key k is stored. These replicas lock themself for the key and replies to coordinator with their versions.
- Coordinator waits to receive reply/version from at least Qw replicas and calculates a new version number and send write to these locked nodes.
- The nodes then write into their database (keeps track of previous value) and replies back to coordinator.
- Coordinator then after receiving reply from all, send release lock message.
- There is a timeout at the locked nodes, after which they roll back the key value.

Handling Crash Failure :

We handle crash failure of a node..

- If a node in unsuccessful in collecting the quorum, it indicates a crash fault, which means we can now look for read quorum-1 (or write quorum -1 for writing).

Failure Detection:

Visualize the structure as the ring of N nodes. (All the nodes know the structure)

Each node send heartbeats to the clockwise next node.

The crash failure of the node is detected by the next node when it doesn't receive heartbeat for timeout time after which it initiates crash recovery.

Crash Recovery

The 'ownership' of the data of crashed node goes to next live node -in this case 'recovery coordinator'

The data is distributed such that the invariant below is maintained :

> "Each key-value pair is stored at an owner and subsequent 2 live nodes."

References

1) Cassandra - A Decentralized Structured Storage System, Avinash Lakshman, Prashant Malik
2) https://www.datastax.com/resources/tutorials/partitioning-and-replication
3) https://dzone.com/articles/introduction-apache-cassandras
4) https://github.com/apache/cassandra
5) https://www.javatpoint.com/nosql-databases
6) https://www.mongodb.com/scale/nosql-database-implementation