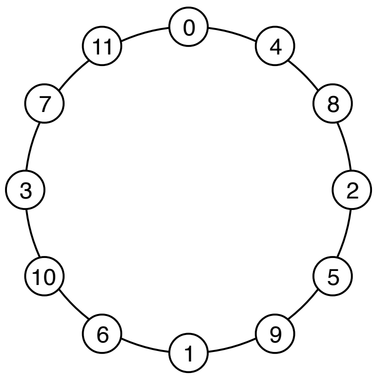
CS 677 Project 1: Distributed File System Doc(v 1.1)

Nodes in hash ring rule

I will fix the location of first 12 nodes. After 12 nodes, new nodes will be randomly placed in the ring.

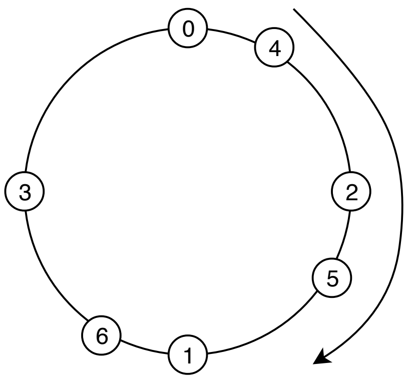


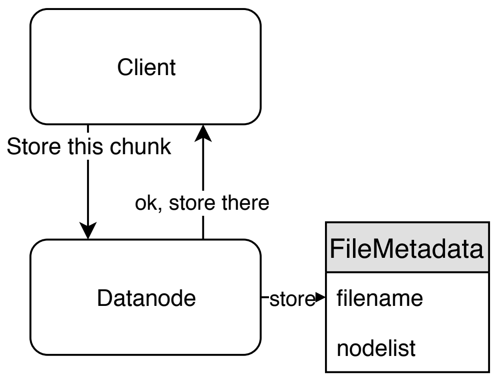
The number above is the order that new nodes will be placed. If only two node 0,1. Node 1 will manage (0,2^159] hash numbers while Node 0 will handle with (2^159,2^160].
If all 12 nodes are on-line, each one will handle 2^160/12.

Store data rule

Using P2P. First, client contact with one node(the first node will be fixed) where to store. This node will use SHA-1, place the file into the hash ring. Then create a node list and return this list to the client. The rest chunks will use this list to store chunks in each node one by one.

This list will also store in the metadata map in the node, and broadcast to all other nodes.



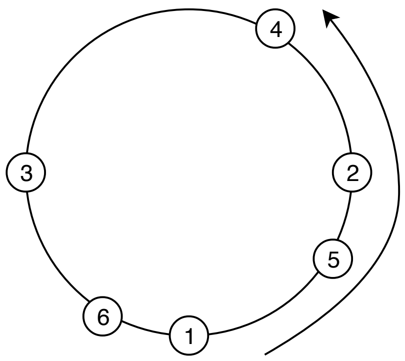


Download data rule

The client will ask a node for the list of nodes who contains the file. The node will load the list from file metadata map and return the client.

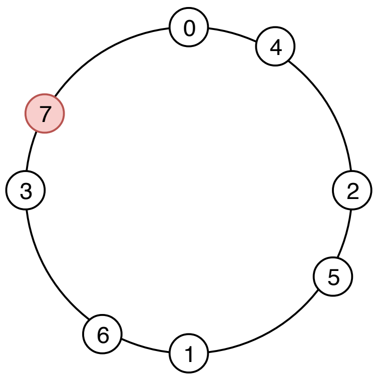
Replicate data rule

Unlike storing data, replicate data will go anti-clockwise, if the original chunk stores in 5, it will have two replications in nodes 2 and 4 to make sure total number of replication is three.



Adding Nodes into hash ring

While adding new nodes into hash ring, the range of nodes will change, which means some nodes will place their local data to the new node as replication; Some node need to delete replications since the number of total replication exceed three. My algorithm is below:

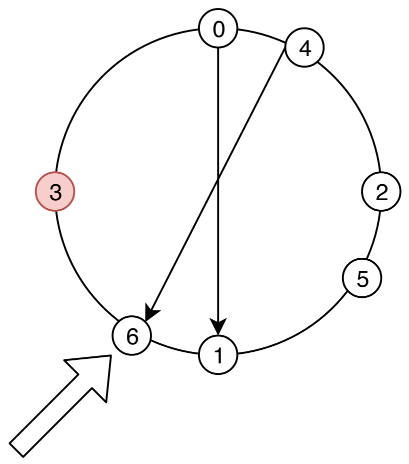


Node 7 is a new node. Before node 7 being added, node 3,6 handle with node 0's replication; node 0,3 handle with node 4's replication. Now node 7 will handle with node 4's replication, node 3 need to remove node 4's replication. Node 6 also need to remove node 0's replication as well.
Coordinator will ask node 0, 4 to replicate local chunks into node 7, at the same time, it asks node 3 to remove node 4's replication; node 6 to remove node 0's replication.

Remove bad nodes

Good nodes will always send heart-beat messages to the coordinator every three second. If coordinator cannot receive heart-beat from a node, it will mark this node as a bad node and remove it.

Once node fails, it means several chunks cannot meet the required number of replication. So the ring need to solve this problem.



In this case, node 3 is a bad node. The coordinator has detected and removed it from the ring. Then coordinator will ask a node before bad node (node 6) to begin repairing replication process.
1. Node 6 will send node 3's local data to node 5, and tell other nodes node3's local data becomes mine, do change in your list.
2. Node 6 ask node 4 to send its local data to itself as replication.
3. Node 6 tell node 1 to download node 0's local data as replication

Coordinator Fails and Recover

The node always sends the newest node list, its usage, and the number of request to the coordinator. if the coordinator fails and lost the node list, once it goes back online, it finds the heartbeat message has a node list greater than itself, the coordinator recognize it has failed, and update the node list to the newest version.

Architecture

1. Data node
2. Coordinator
3. Client

MetaData
hostInfoList: List<String, Integer>
filenameMap: Map<String, List<String, Integer>>
host: String
Port: Integer

DataNode
host: String
port: Integer
hostHashMap: Map<hashHost: Integer, List<String, Integer>>
dataChunkMap: Map<hasHost: Integer, Map<filename,chunkList>>
askLocalChunkFromHost: void
addChunkLocally: void
removeChunkLocally: void
getChunkFile: Byte[]
sendReplication: void
heartBeat: void

Coordinator
host: String
port: Integer
hostHashMap: Map<hashHost: Integer, List<String, Integer>>
hashNewNode: Integer
addHashNodeMap: void
removeHashNodeMap: void
heartBeatManager: void
updateHashNodeMapAll: void

Client
host: String
port: Integer
dataNodeHost: String
dataNodePort: Integer
filenameMap: Map<String, List<Byte[]>>
uploadFile: void
downloadFile: void
splitFile: void
combineFile: void