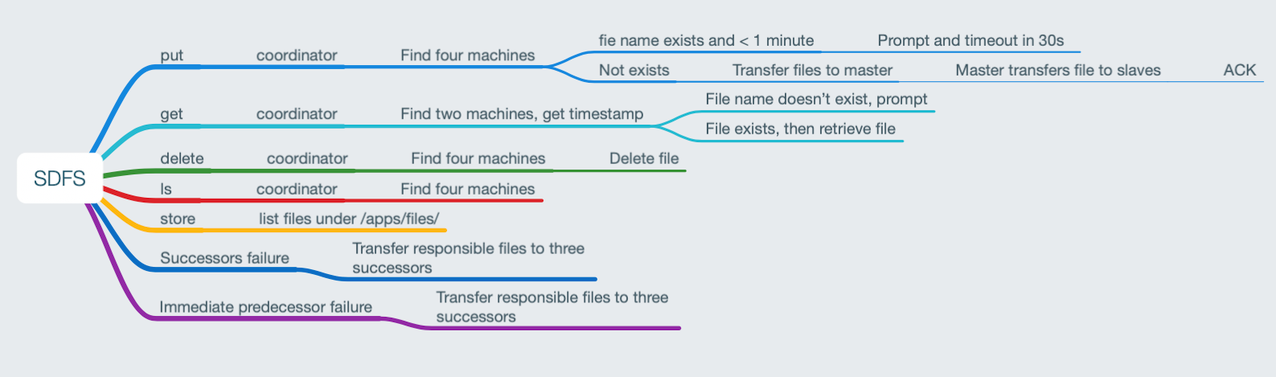
MP3 Report

**Design:**



We used ring structure; each node is hash-partitioned in a ring of 1024 slot. We use address as input to generate hash code and take the modulo of 1024 to determine the slot for each node, 1024 is a constant we choose, it can be arbitrarily large number to scale up this program.   
For each client, we use local node process as coordinator which interacts with other nodes. In our system, any node can be a coordinator to interact with client.  
Each server is a primary replica of file which hash to the server’s ID. We call these file responsible files of this primary server. And its three immediate successor contains backup replicas.  
*PUT: Write*  
When a client send a put request, coordinator will take the sdfsfilename hash it and put it on the corresponding servers. The coordinator will start a TCP connection with all responsible replica and put the file to it with same timestamp. after that, the coordinator waits for ack from quorum. In our assumption (at most three failures), when coordinator receives 3 ack, it tells client the write is done, then the next operations can be scheduled.  
*GET: Read*  
A client send GET request, coordinator will take the sdfsfilename, turn it to hash code and find all servers that contains replica. It will wait for first 3 replica to send its timestamp. And the coordinator will fetch the file from server with latest timestamp.  
*DELETE:*   
coordinator will send delete request to all replicas and confirm all of them successfully deleted the file.  
*LS:*  
coordinator will send LS request to all replicas and see if the file exists, if so, it will send these hostname to client and then print it to console.  
*Failure, Leave and Rejoin:*  
If any server failed or leaved. First, all live servers will get the delete request. Then, every server will start check if its immediate three successor have its replica, if not, it will start a TCP connection and send all responsible files to whoever lack of it. If any server joined, all live servers will also get the join request. Then, we will check if they contains file that it doesn’t need anymore and delete them. They also check if the new joined node is in the three successor range. If yes, it will send the responsible files to this new node.

**Experiments:**

i.  
We first put a 40M dummy file to our SDFS. Then we find out files are stored in vm 3,4,6,10. So we go to vm3 which is the master replica and start monitor TCP bandwidth using iftop. We can see the bandwidth is around 1KBps, which could be our heartbeat message. Then when we kill our process in vm4, we see a surge in bandwidth in vm3, which is 20MBps. Since iftop updates every 2 second, we cannot see but we believe it is probably been 20MBps for 2 seconds which is 40M, our file size.

ii.

here is our experiment result for 500M file and 25M file in all three operations. Our insert and update is using same logic so their execution time is about the same. Read is faster compare to write, because we only need one TCP connection for file in read but four connections in write. We compared the file transfer speed with SCP (7 sec), we are a bit slower than scp which took 4 sec, but since we are sending 4 files concurrently, we think we did better overall. We can also see 25M file writes much faster than 500M files.

iii:

We have very similar performance for 4 VMs and 8 VMs on storing Wikipedia data into SDFS. In our design, we will wait for each file to finish write then start the next file, so the bottleneck is our coordinator, it reads file from client and send it out to four replicas. We believe this is the reason they are about the same speed.