MP3 Report

Jun Luo junluo2
Ruochen rs20

Design:

The overall design of MP3 is similar to the MP2. Like sending heartbeats to 3 immediate successors in MP2, we have a master node for each file and the master node sends replicas to its 3 immediate successors. The way to select the master node for a file is the same as the Chord protocol.

We used the ring structure; each node's ID is hash-partitioned in a ring of 1024 slot. We use the address as input to generate a hash code and take the modulo of 1024 to determine the slot for each node, 1024 is a constant we choose, it can be an arbitrarily large number to scale up this program.

There are two roles in our design: **client** and **node**. Each VM runs a node and we can use the client to interact with nodes. The client preprocesses the command typed by users and sends the post-processed command to the node on the same VM. We call the node on the same VM to receive commands from the client as **coordinator**. Every node maintains a full membership list for join and failure (MP2) and manages replicas for files it owns. When there is a change in the membership list, a master node will try to duplicate replicas or transfer ownership for certain files.

We offer following commands for the client and there are brief descriptions for each command.

Put <localpath> <sdfaname>:

The client sends the "put" request to the coordinator, the coordinator will hash sdfsname and put it on the corresponding servers. After receiving quorum (3) acks, the coordinator tells the client the write is done.

Get <sdfaname> <localpath>:

The client sends the "get" request to the coordinator, the coordinator will hash sdfsname and find all servers that contain replica. It will wait for the first 2 replicas to send its timestamp. And the coordinator will fetch the file from the server with the latest timestamp.

Delete <sdfaname>:

The client sends the "delete" request to the coordinator, the coordinator will send delete requests to all replicas and confirm all of them successfully deleted the file.

Ls <sdfaname>:

The client sends the "ls" request to the coordinator, the coordinator will send ls request to all replicas and see if the file exists. If so, it will send these hostnames to the client and then print it to the console.

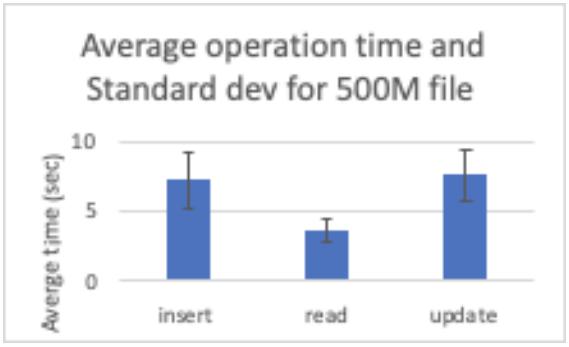
Store:

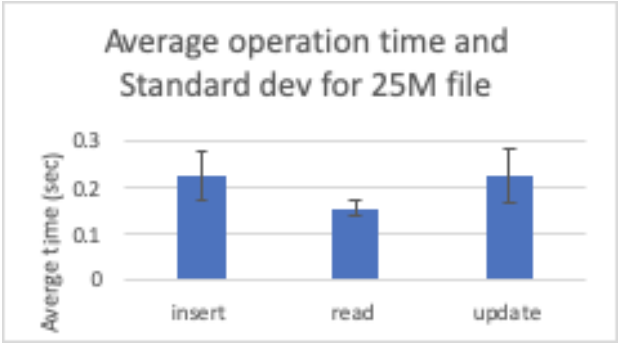
The client prints all files under a specific folder that stores all SDFS files in the VM.

Failure and Rejoin:
If any server failed or left, all servers will receive a message due to MP2 logic. Then they will start to duplicate by checking if its immediate three successors have its replica. If not, it will start a TCP connection and send all responsible files to whoever misses files. If any server joined, all live servers will also get the join request due to MP2 logic. Then they will check if they contain files that it doesn't need anymore and delete them. For the newly joined node, all alive nodes also check if the new joined node is in their three successor range. If yes, they will send all necessary files to this new node. Also, Files will be updated if their master node ID changes during leave or join. If the immediate predecessor changes, a node may transfer ownership for certain files.

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. Our re-replication time is about 0.4 second which is very close to put a file with 40M.

ii. Here is our experiment result for a 500M file and a 25M file in all three operations. Our insert and update is using the same logic thus their execution time is about the same. Read is faster compared to write, because we only need one TCP connection for the file when read but four connections during write. We compared the file transfer speed with "scp", it took 4 seconds and our program is a bit slower since it took 7 seconds. However, since we are sending 4 files concurrently, we think we did better overall. We can see 25M file reads and writes much faster than 500M files which is expected.





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 writing then start the next file, so the bottleneck is our coordinator, it reads the file and sends it out to four replicas. We believe this is the reason they are about the same speed.