CS425 MP3 - group 31

bmjain2 (Bhavana Jain) and dipayan2 (Dipayan Mukherjee)

We have built a distributed file system (SDFS) on top of the membership group of MP2. We have a centralized master server that stores all the meta-data about files and nodes in the system and hence all queries go through it. We replicate a file on 4 nodes to provide availability against three simultaneous failures. Our system can tolerate master node failure.

Components of the design -

- fileMap: file to list of node ids mapping on master
- nodeMap: node to list of files mapping on master
- fileTimeMap: Each member node stores a map of its files and the last updated timestamp.
- conflictMap: stores the timestamp of the last put command executed for a file. This is used to detect write-write conflicts.

Event protocols:

- **put localfile sdfsfile:** Send a put sdfsfile request to the master. The master checks conflictMap to see if the last put on sdfsfile was within 1-minute of this request. If yes, handle conflict. If successful, it replies with a list of nodes -- selected randomly for insert and retrieved from fileMap for update. The put initiator sends this file to the nodes which then write it in their temp directory and send an ACK upon receiving the entire file. Once the put initiator gets all the 4 ACKs, quorum is reached. It notifies the master which then sends a message to these nodes to move the file to shared directory.
- **get sdfsfile localfile:** We send a get request to the master, which sends us the list of nodes that store the sdfsfile. Note that all the nodes have the latest version of this file (write requires a quorum of all 4 nodes). We loop through the nodes and request the file -- once successful, break out of the loop.
- **delete sdfsfile**: We forward the delete request to the master, which checks the fileMap and finds the list of nodes which store the sdfsfile. It sends a deletefile command to these nodes which then remove the file from the shared directory.
- **ls sdfsfile**: Send a get request to the master to receive the list of nodes which store the sdfsfile and output.
- **store**: Output the files present in the fileTimeMap.

Failure handling:

- **Master:** When the master is detected as crashed, each member node starts a leader election (ring protocol). Member node with the smallest ID declares itself as the leader, and disseminates elected leader messages. When the other nodes receive the elected leader message, they send information about all the sdfsfiles they're storing to the new master. This way the new master learns all the meta-data about the system.

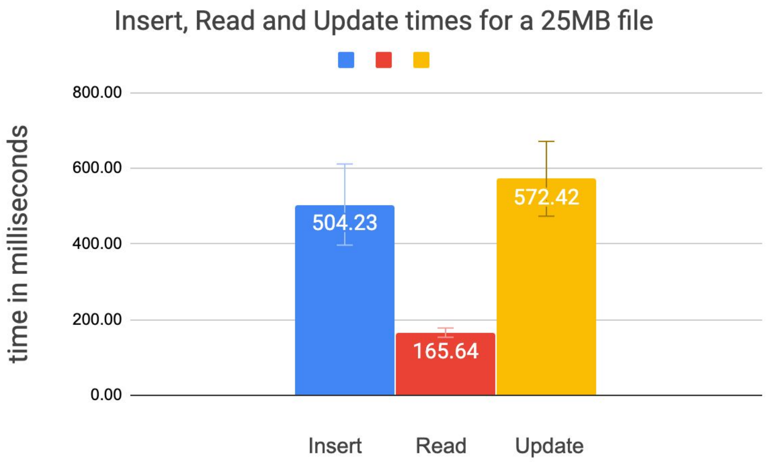
- **Node:** When the master detects failure of a node, it starts an active replication of all the files stored by the crashed node.

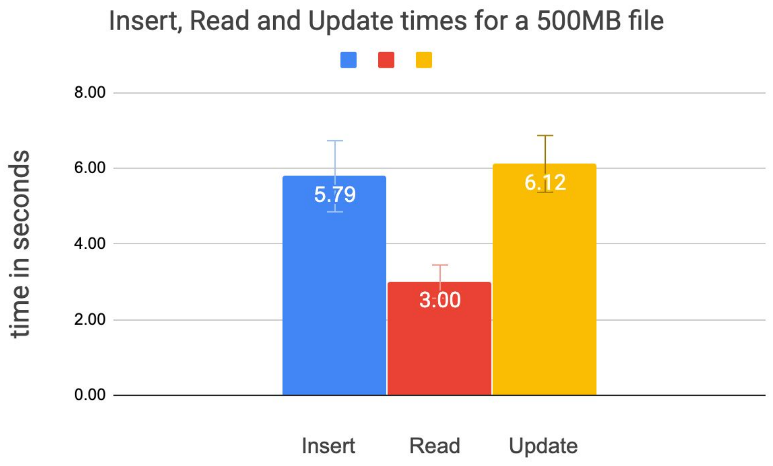
Background Functions:

- **HandleFileReplication:** Master periodically checks the file replication count of each file in the system and if it is less than 4, it asks a node (which has the file) to replicate the file to other nodes. (passive replication)

Debugging using MP1: We log all the events -- put, get, replicate and MP2 events, steps involved, errors etc and store them in a logfile. We start the MP1 servers at each node in the system and perform a distributed grep using the MP1 client.

Measurements:

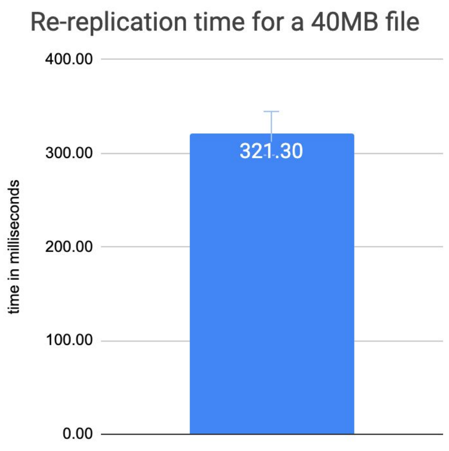


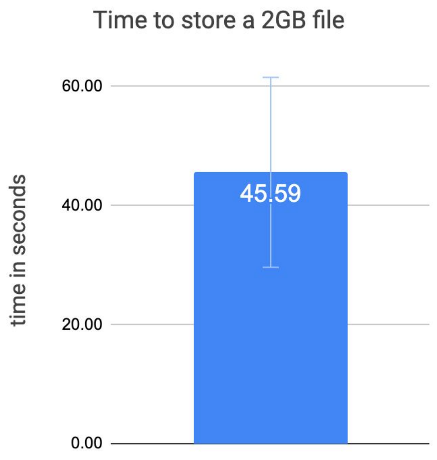


i) Re-replication of a 40MB file takes about ~320milliseconds to replicate (std-dev of 23milliseconds) and an outgoing bandwidth of 82B/s

ii) The insert and update time is higher than the read time. A read gets the list of nodes from master and simply fetches the file. However, for insert and update, a quorum of 4 nodes has to be reached. This requires ACKs from all node -> put initiator -> master -> send confirmation to all nodes. Erratic network latencies among nodes lead to large deviations in the insert and update times.

iii) Our system does not shard an incoming file and hence it wouldn't matter if we used 4 machines or 8 machines. Our latencies in both the cases are similar. We generated a random 2GB file to test this. It takes about ~46seconds (std-dev of ~16seconds) to store the file.


Re-replication time for a 40MB file
time in milliseconds


Time to store a 2GB file
time in seconds