baohez2(Baohe Zhang) and kechenl3(Kechen Lu)

CS425 MP3 Report Group 29

Part I System Design:

Our simple distributed file system(SDFS) implemented a HDFS-like distributed file system, which includes the versioned file control, fast file replication mechanism, master node re-election and data node re-replication after failures, and leverages the membership service and failure detector implemented at last MP.

In our SDFS, we have a master node and multiple data nodes. A node running as a master node could also work as the data node. Master node plays a role as the coordinator like Cassandra, that the client contacts directly with it, handling the operations or commands initiated by the client. But the master node would not handle everything. It only decides where SDFS file replicas should be placed in which datanodes and maintains a metadata(a more complicated hashmap) of SDFS files.

For Put command initiated by client, in our SDFS, it leverages the replication pipelining of the HDFS. When a client run a put command, it first contacts the master node, while the master node hashes the sdfsfile, generates a timestamp, decides which data nodes keep the replica by hashing to get a consistent range of replication, and return all of these information to the client. As the instruction of spec, we specify each file version has four replicas kept in four data nodes for the at most three machine failures because the master node has the up-to-date metadata and W=4, R=1 is enough for a client to do the read latest written file. When client receives the meta info from master node, it would send one write request and the file to one data node. If the data node receives, it stores replica persistently in local while simultaneously transfer this replica to the next node responsible for this replica. This mechanism averages the bandwidth of each node and the client, and could attain a high speed of file replication. The replication pipelining shows as below Fig 1.

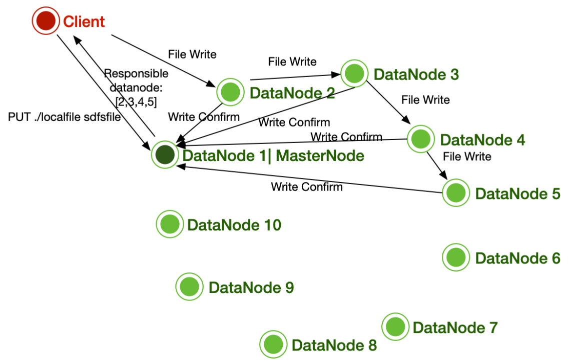


Fig. 1 DataNode Replication Pipelining

For Get command, the client would send Get request to master node and master node returns a list of data nodes who have this file with latest version, then client iteratively read from one node and at least and mostly only read once. For Delete command, after receiving the Delete request from the client, master removes the file info from its metadata and notifies data nodes responsible for this file to remove all versions and info in its “partial metadata”. For ls and store command, the client only queries the master node to get all nodes info or files in one node from its metadata.

For versioned file control, We can assume that all nodes running with the synchronous time. So we use the timestamp when a master receiving a file version put as the version number ,and all updates of this given file delivered would be in the same order among all nodes keeping this replica.

baohez2(Baohe Zhang) and kechenl3(Kechen Lu)

CS425 MP3 Report Group 29

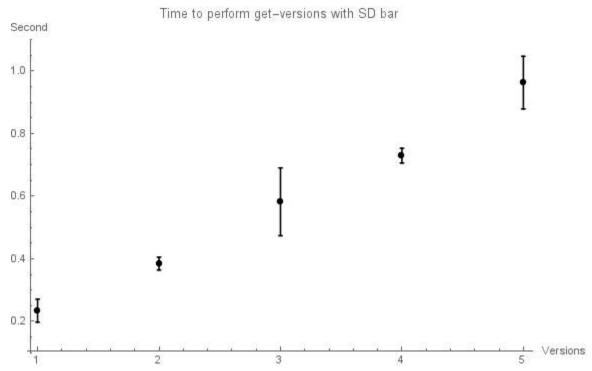
In the SDFS, we leveraged the code from the last MP, which implemented a SWIM-like membership and failure detectors. When a failure happens, the membership service would notify master node through Go channel and then re-replication mechanism would be initiated by the master node. Master node would iterate its metadata, and find if a file version has a responsible data node not in the membership list, replace it with another data node randomly picked. After this, master send out re-replica request starting with the node having the replica and also pipeline it with a new data node list after replacing some failure nodes.

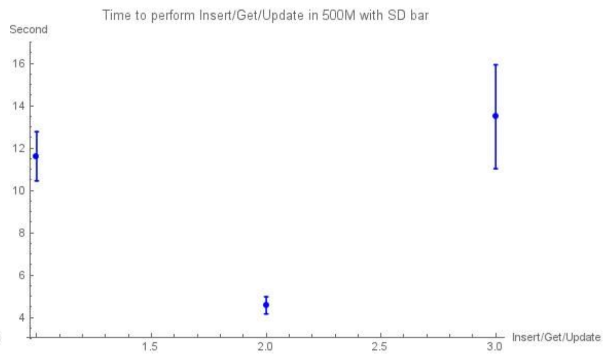
For the master election after its failure, we use bully algorithm to elect a new master. A node initiate an election when it detects the master's failure. The node which has highest IP address will finally win the election and become the new master. The new master is also served as the membership system's introducer.

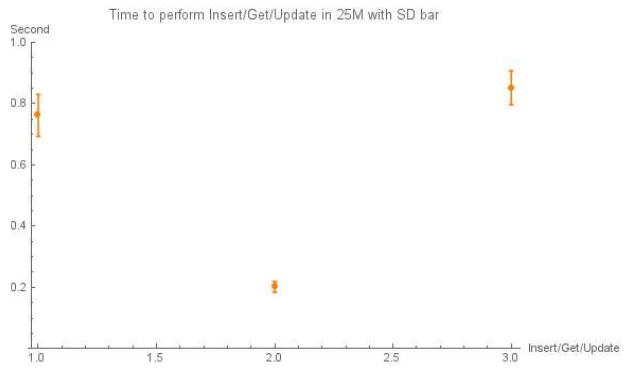
For using distributed grep in debugging SDFS, it helps a lot to explore the correctness of file replication and re-replication in data node and master re-election after failures, and then optimize our implementation like re-replication time.

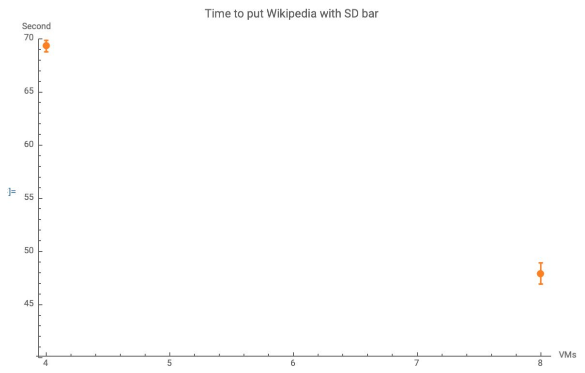
Part II Evaluation:

(i) When a node fails, the peak bandwidth of re-replica sender is 8.36MB/second, while the replica file size is 40MB. Re-replication time is around 5.63s.









(ii) The time of get and put command is linear with the filesize. In the cluster environment, the put speed is about 45 M/s, the get speed is about 125 M/s.

(iii) The time of get-version command is linear with the version number, that because we put a file's all versions in the same set of nodes.

(iv) Puting wikicorpus into 4 machines takes about 69 seconds, into 8 machines takes about 47 seconds. One possible reason is that since we use replica pipeline, the file transfer speed is dependent on the slowest link of two consecutive nodes.