Yinfang Chen, yinfang3@illinois.edu
Ragini Gupta, raginig2@illinois.edu
CS 425
Fall 2022

Simple Distributed File System

Algorithm Design:

For SDFS, we implement a master-followers architecture where Machine-01 serves as the introducer by default that interacts with the user. Next, a coordinator node is elected which serves as the master node and takes care of re-replication of the files in case of failures as well. The master node for a file is computed by computing the hashed value of the file. The replication factor for a file is 4 in the system so 4 copies of a file will be stored across the cluster (including 1 master/coordinator node and the rest replicas on the follower nodes). Consistent hashing based mechanism is used for maintaining the replicas in the cluster. If any of the replica dies, a new replica node is selected by the master node using consistent hashing from the available live nodes. For example, if node-1 is the master node, node-2,3,4 will contain the replicas. If node-4 fails, a new node is selected to re-replicate the file to a new node selected after applying consistent hashing. Similarly master node is responsible for file shuffling and rearrangement when a node joins as well. Each node also maintains a data structure (fmapper.py) storing metadata for all sdfs data files stored across the nodes along with timestamp. We use a practical timeout interval to detect the failure to decrease the false positive, and we consider cpu efficiency when implementing the failure detection.

Each of the functions are implemented as follows:

For PUT function: Load a file with local file name and store across sdfs nodes. The node checks IP of master node first and it's replica nodes based on the hashed value of the file and then it will transfer the file to the master and its corresponding slaves to store the file to the sdfs file system. TCP socket connection is used for communication.

For GET: On receiving the GET command, the query node first checks if the file exists and then fetches the file using socket TCP communication. Read Quorom R=1 is used for reading/get operation.

For DELETE: To handle delete operation, Quorum W=4 is used to ensure delete from all nodes containing the file. The query node checks for the file and asks all servers containing this file to delete it.

MP-2 was used for Failure detection in the cluster. It was highly useful to replicate the files across the nodes present in the membership list of the master. MP-1 is used for log mining and debugging the code.

Re-replication time and bandwidth upon failure:

We calculate the re-replication time and bandwidth for 40MB file. The re-replication time involves three steps; detecting a failure, selecting a new node for replication, and re-replicating (transferring) the file to the newly selected node. 0.081 seconds (with a STDEV of 0.042). The bandwidth is: payload size/avg. time_elapsed= 493.82MBps => 0.49GBps (standard deviation of 0.13)

Time to insert, read and update file:

	Insert	**Read**	**Update**

Yinfang Chen, yinfang3@illinois.edu
Ragini Gupta, raginig2@illinois.edu
CS 425
Fall 2022

	25MB	500MB	25MB	500MB	25MB	500MB
	0.063	0.328	0.0066	0.20	0.065	0.38
	0.057	0.49	0.0046	0.190291	0.073	0.389
	0.055	0.31	0.0048	0.180192	0.078	0.46
	0.055	0.38	0.0059	0.16	0.077	0.48
	0.059	0.36	0.0056	0.13	0.069	0.44
Avg.	0.0578	0.37378	0.0055	0.172	0.0724	0.4289
Standard Deviation	0.003347	0.070388	0.000818	0.0278	0.00545	0.0432

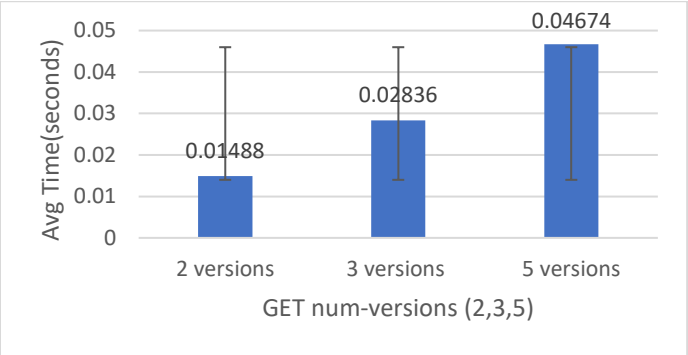
As evident from the above table, Insert and Update operation take nearly the same amount of time as both operations are same. In case of insert operation, the coordinator node will perform consistent hashing to extract the replica IPs of the nodes whereas in update operation it will have to search for the already assigned IPs from the metadata. The reason why update operation might be slightly higher than the insert operation could be that for update the replica nodes have to be extracted followed by replacing the already existing file with the latest version. As expected, the read operation takes the lowest about of time, infact it is almost 9x faster than the insert operation. This is expected because no file transfer is involved in the read operation, the coordinator just has to read the file from one of the replica nodes.

c) Time to perform get-versions using num-versions:

	Num-versions=2	Num-versions=3	Num-versions=5
1	0.0142	0.0259	0.0482
2	0.0156	0.030	0.0459
3	0.0139	0.0318	0.0433
4	0.0144	0.0296	0.0484
5	0.0163	0.0245	0.0479
Avg.	0.01488	0.02836	0.04674
Std deviation:	0.001023	0.003042	0.002166

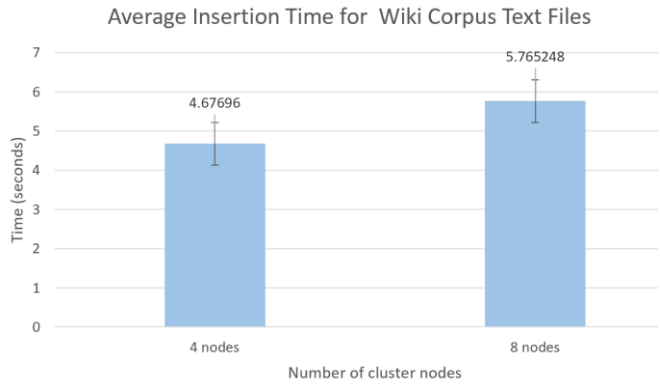
As shown in the above table, as we get more number of versions for a file, the time increases. This is expected because to fetch more versions of the file the delay for file transfer will increase. It is worth mentioning that while extracting the 2 versions it only fetches the latest two versions of the sdfs file, for getting 3 versions it fetches the latest three versions of the sdfs file and for getting 5 versions it fetches all the last 5 versions of the file.

Yinfang Chen, yinfang3@illinois.edu
Ragini Gupta, raginig2@illinois.edu
CS 425
Fall 2022



d) Time to store Wikipedia Corpus text files:

	4 Machines	8 Machines
1	4.981	5.89
2	4.762	5.16
3	5.983	5.99
4	5.10	5.78
5	4.191	6.01
Avg.	4.68	5.765
STD.	0.408	0.351



Since we are implementing the replication strategy based on consistent hashing, the average time elapsed for inserting files on four machines and eight machines is almost similar with a slightly higher latency for eight machines as shown in the graph.

Gitlab: https://gitlab.engr.illinois.edu/yinfang3/cs425-mp/-/tree/main/mp3