CS425, Distributed Systems: Fall 2018 (vandana2, lavisha2)
Machine Programming 3 – Simple Distributed File System

DESIGN

We have implemented the following commands

1. put localfilename sdfsfilename
   We choose VM10 initially to be the leader that is contacted by the client when this command runs (This handles total ordering). The leader maintains a dictionary that contains keys that are sdfs files and values that contains address of machines that store the files and the number of versions for that sdfs file. The leader chooses 4 nodes that are alive in a random manner from its membership list as replicas and inserts the sdfs file on the replicas.
2. get sdfsfilename localfilename
   We contact the leader to fetch the sdfsfile from the replica. The leader selects one of the 4 replicas, fetches the file from the sdfs file system and inserts it into the local file system. The address of the replicas are fetched from the dictionary that the leader maintains
3. delete sdfsfilename
   The client contacts the leader to delete the file from the sdfs file system
4. ls sdfsfilename
   The client contacts the leader to fetch the list of all the machine addresses where this file is present. The leader makes use of the dictionary, searches for the key that stores the file names. It then fetches the value of the dictionary that contains the list of VMs that stores the file in its sdfs file system
5. Store
   Store executes ls command on its local files system and fetches the list of all the files stored.

When the leader fails, we have adopted ring based leader election algorithm to select the next leader. Hence the leader with the next highest ID is chosen as the leader.

Failure Detection - Once the leader detects a failure of a node, it check its dictionary to determine the files stored on the node that failed. It then replicates the file onto a machine that does not already store the file. It then goes and updates its dictionary regarding the information of the new node that will be the new replica.
We do not make use of any quorum since number of writes is equal to 4 and number of reads is equal to 1. Since W+R (=4+1)= 5 is greater than 4 (number of replicas), read and write consistency is maintained. We use MP2's code to access the membership list, to fetch the nodes that are alive for inserting the replicas. We also used MP1 for identifying errors for checking the debug output.

(i) Replication

Average time - 31.9709 seconds
Standard Deviation - 3.789 seconds
Bandwidth: Receiving bandwidth - 80.66kB/s; Transmitting bandwidth - 42.77 MB/s

For transfering a 40MB file as expected the bandwidth is a little more than the size of the file at the leader node, while the receiving bandwidth is negligible in comparison.

(ii) 25MB

Average Time

Time to insert - 4.324 seconds; Time to read - 4.36 seconds; Time to update - 3.724 seconds

Standard Deviation Time

Time to insert - 0.283 seconds;Time to read - 0.310 seconds; Time to update - 0.298 seconds

500MB

Average Time

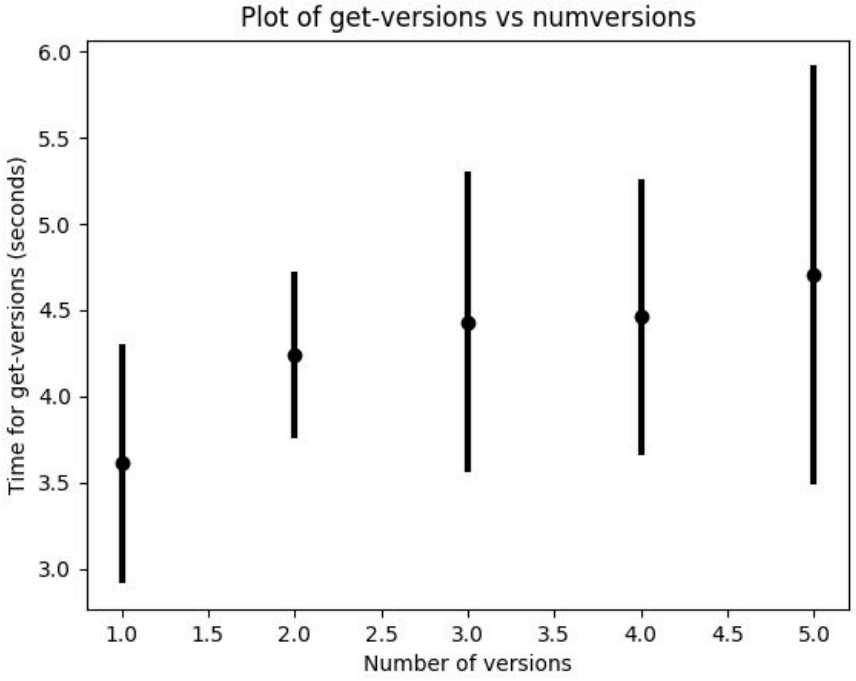
Time to insert-30.429 sec ;Time to read - 31.36 sec ;Time to update - 29.903 sec

Standard Deviation Time

Time to insert - 1.053 second; Time to read - 0.981 seconds; Time to update - 0.879 seconds

As expected, the time to insert the file increases linearly with increase in the file size. However since the number of replicas remain constant, the time to update remains constant.

(iii) The dots represent the average time while the bars represent the standard deviations.



As expected, the average time to fetch the files increases with the number of versions, though the increase is steeper in the beginning and reduces later. The std also increases with more versions as more data is being fetched.

(iv) 8 machines

Average time : 127.8 seconds

Std-Dev: 7.24 seconds

4 machines

Average time : 110.8 seconds

Std-Dev: 5.59 seconds

Due to constant number of replicas, the average time seems to remain constant. However the standard deviation increases with increase in the number of machines. This might be due to increase in the network bandwidth