MP3 Report

Fangwei Gao, Yuyao Wang
fangwei2@illinois.edu, yuyaow3@illinois.edu

Design

Since the SDFS is tolerant up to 3 machine failures at a time, the number of replicas should be 4. We use R = 1/W = 4 as our Quorum consistency level to make sure that get operation always reads the latest value. We have a master node to keep all the file information organized and the master node also gets involved in many related file operations. Each member node has its own local file meta-information map and master possess a file meta-information for the whole system.

Leader election: we used Bully algorithm for leader election. We set introducer as a default master node first. When the master node fails, the nodes that detect this failure will launch the election process, which eventually picks up the node that has the highest host id value as the new master. The new master will send its elected message to all members.

Put: First, the node that handles the put queries the master for the file version. Then, this node saves the file locally, sends 'put' message to the master and gets 3 random replicas from the master node. Finally, it sends files directly to the 3 replicas. Put finishes after all the replica nodes acks back. Notice there are 4 replicas stored in the system (the node itself and 3 other random members).

Get/Get-versions: To read a file: first, check whether itself stores the requested file locally. If yes, it immediately sends that file back. Otherwise, that node queries master for the replica nodes information (nodes that stored the latest version of the file). Then, the node sends 'get' message to one of the replicas. Finally, the node one who receives 'get' request sends files to the requested node. Since we use the W=4/R=1 Quorum, get will always get the latest version of the requested file quickly. Get-versions works similar as getting. Instead of asking the master node for the latest version, it asks master for the replica locations of the requested versions then gets the files back from those locations.

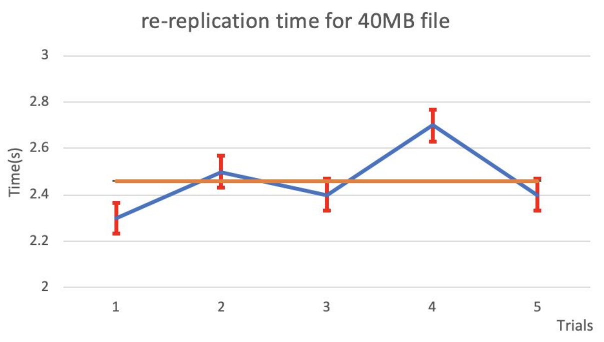
Delete: A node sends 'delete' request to master first, master check whether the file exists in the meta-information map first. If the file exists, the master sends 'delete' request to all the replicas that have the file, the replica receives the request will remove the file from its local and respond to master. Finally master sends the result back to the node that asks for delete.

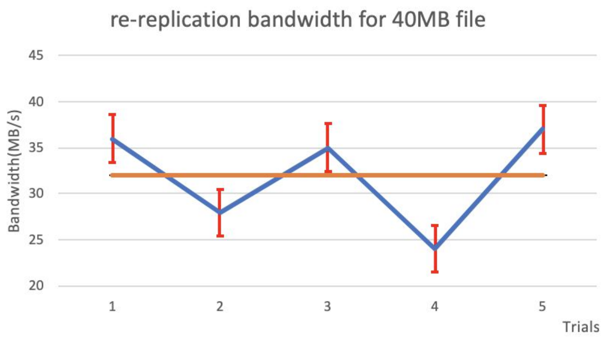
Node fail: we use our MP2 as the membership system and failure detector of our SDFS. In order to backup the master node meta-information, master periodically sends its meta-information to randomly picked 3 members. When non-master node fails, the nodes that detect the failure will inform master and master then asks other alive nodes to restore the files of the failure nodes. When the master node fails, election process will pick up a new master and the new master will ask all alive nodes for previously backup meta-information. It will pick the one with the latest timestamp as it master meta-information. As soon as it gets all the information needed it will start to ask other alive nodes to restore the files of the failure nodes just like before.
MP1 helps us in debug with logs. We write logs into a local log file and query remotely for the information we need. MP2 also works as the membership system and failure detector for our file system.

Measurement

1. re-replication time and bandwidth upon a failure:

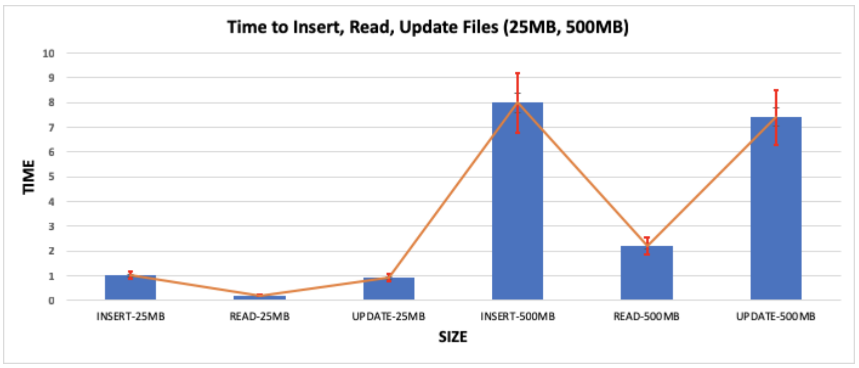
It takes an average value about 2.5s for re-replication process for a 40MB file when a failure happens. The standard deviation is large because the value relies on network status. The two graphs are negatively correlated. Since the system will generate almost the same amount of messages for the same node failure of different times. When re-replication works fast, the bandwidth goes up.





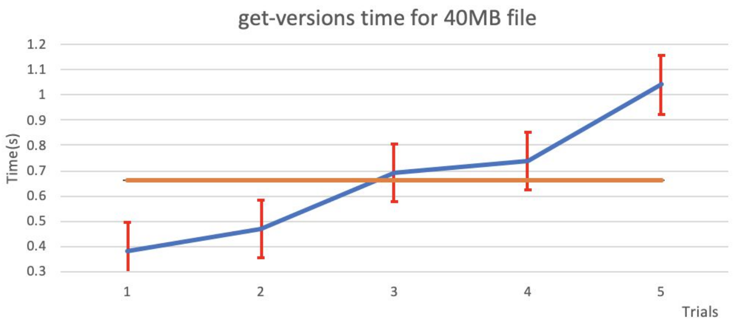
2. *times to insert, read, and update, file of size 25 MB, 500 MB, under no failure: time(s)*

Insert a 25MB file is absolutely faster than insert a 500MB file. In our design, read is much faster than write and upload because of W=4 (4 replicas) and R = 1. But many uncertainties also get involved, such as the node could have the file stored locally, which gives a much faster read. Also, when the file size gets large, the insert, update and read performance vary significantly.



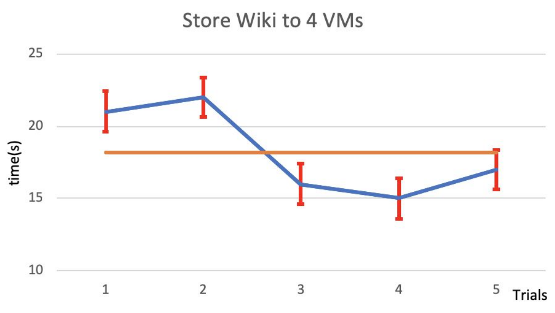
3. *time to perform get-versions as a function of num-versions:*

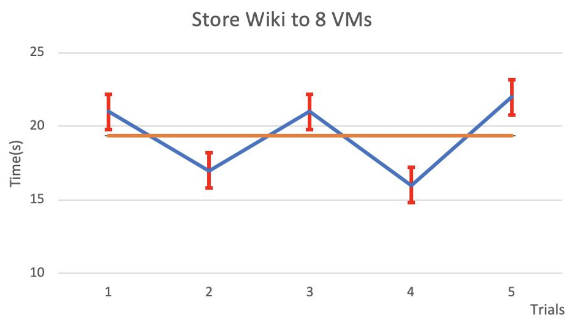
The time spent and the versions requested are almost linearly correlated. Because the data a node needs to send and read is linearly proportional to the versions of file it is requested.



4. *time to store the entire English Wikipedia corpus into SDFS with 4 VMs and 8 VMs*

Store a large file (Wikipedia 1.35G) takes around for 4 VMs take about 18s and for 8 VMs take about 19s. The number of vms does not influence the performance much. But the performance is not stable due to unknown network condition.


Store Wiki to 4 VMs
time(s)
Trials


Store Wiki to 8 VMs
Time(s)
Trials