CS 425 Distributed System MP3 Report

Group 43: Yeongyoon Park (ypark66), Tzu-Bin Yan (tbyan2)

I. Design

In this MP, the design of the SDFS is mainly composed of 5 parts: the UI thread, master thread, slave thread and the membership list component from the MP2 that resides in the SDFS server program, as well as the client program itself.

For the SDFS server program, the UI thread is the main thread of control that starts the failure detector and slave thread based on the pass commands. The slave thread is a thread that is always present on all VMs running the SDFS, and is in charge of managing replicas (including storing, sending & deleting a replica) that are currently stored locally based on the master thread's commands, as well as coming up with the master of the SDFS by performing election among all slaves. The master thread is a thread that is only run in the same process as the slave who won the latest round of election, and is in charge of performing operations based on requests sent by clients (including quorum-based write-write-conflict-supported putting, quorum-based getting and deleting files from SDFS as well as listing all VMs that has a replica of a file) and ensuring that the replication of the files is compliant toward tolerating three simultaneous machines failure at a time. The membership list component is the same as in MP2, where it sends heartbeat to some node in the system, detects the failure of some nodes in the system, sends out the failure detection results, and updates the membership list accordingly. All communication between master, slaves and clients use TCP as a reliable way of communicating (the membership list component still uses UDP).

The election algorithm used by the slaves to elect the master is the ring election algorithm, where the slave having the highest ID (as in dictionary order) among all in the SDFS will be elected, who will then in turn start the master thread locally to serve the clients.

The replication strategy used is passive replication, where the master acts as the replica manager for replication and performs replication based on requests by clients. The replica that is stored at each slave is merely a full copy of the original file to reduce the complexity of the SDFS design. A total of 4 replicas per file will be present in the SDFS (at different slaves) to tolerate three simultaneous machines failure. The quorum size that is used in put and get commands to have put and get be fast is simply $4 / 2 + 1 = 3$.

For the client program, it performs most of the commands (except store, where the client directly contacts the slaves) by first asking the slaves for the master's address and then send the request (constructed based on command line inputs) directly to the master for processing, who then deals with the request and may contact other slaves in the system. Based the messages sent back by the master, the client will perform the 60 second write-write conflict prompt to notify the user of the event and ask for explicit consent to proceed the "put" command, which will be timed out after 30 seconds if no response is given by the user.

MP1 is useful in this MP since during debugging, we can use the remote grep functionality to check if the file operation issued by the master is indeed performed by the slave without having to directly ssh into the each VMs one by one to check if it that's indeed the case, which is very time-consuming.

II. Measurements

(i) Measurement Setting:

One machine failure (storing a replica of a 40MB file) in a SDFS with five machines, where the master will then request a machine not storing the replica to retrieve the replica from another slave storing a replica.

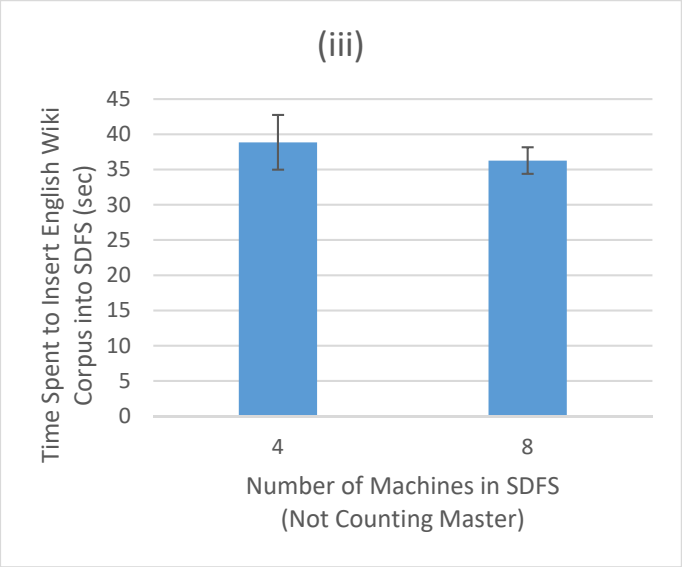
Average re-replication time: **0.2143** (sec)

Sample STD for re-replication time: **0.008411** (sec)

Average bandwidth per machine measured over 30 second interval: **366948.9467** (Bps)

Sample STD for bandwidth: **451.5526** (Bps)

The measurements of (ii) and (iii) are shown below in plots. The measurement setting for (ii) is a SDFS (5 machines) with a client on one of the machine issuing the requests ("get" and "put"). The measurement setting for (iii) is the specified number of machines in a SDFS while a client on one of the machines "put" the English Wiki corpus (not already present in the SDFS) into the SDFS



For the average re-replication time, we can see that the time needed is relatively low. This is due to the file being transmitted being rather small, and that based on the replication design, the replica is only transmitted once during the re-replication (from one of the slave holding the file to another slave that is requesting the file based on the command from master).

For the average time spent on each operation (as shown in graph (ii)), we can see that the insert operation (as done when a file is first "put" into the SDFS) always takes more time than both read ("get") and update ("put" when the file is already in SDFS) operation. The reason for this is that the quorum design in both read and update allows the master to acknowledge the client before performing read or update operations on all slaves storing the target replica, while the insert operation is designed to acknowledge the client only after all required replicas are stored at the respective slaves. As for the reason why the read operation spends the least amount of time among the three, it is that the file transmission time incurred in read operation is much less than that incurred in the other two, since the target file is only required to be transmitted once during the read operation (i.e. from a slave to the client), while in both insert and update the target file is required to be transmitted at least quorum number of times by the master before the acking the client.

For the average time spent on inserting the whole English Wiki corpus into SDFS with different number of machines (as shown in graph (iii)), the average time spent w.r.t. different number of machines in the SDFS is roughly the same. The reason for this is that in our SDFS design, the number of replicas stored in the SDFS is independent of the number of machines present in the SDFS (i.e. as long as there are at least four machines in the SDFS, we have that only four replicas will be present in the SDFS no matter how many machines there are), therefore the insert operation will always require the file to be transferred the same number of times in both scenarios and thus will take about the same time (file transmission time is dominant for English Wiki corpus insertion since it is very large).