Team 55 (yixin10, yangt2) – MP 3 Report

1. Design

1.1 Overall Algorithm Design

Since there are at most three simultaneous machine failures, we have **four** replicas for each file, which means there are always four copies for each file in SDFS. Also, since any two conflicting writes intersect in at least one replica, and any write and read intersect in at least one replica, it follows that W+W > 4, and W + R > 4. Therefore, to make W+R the least, **W=3, R=2.**
In our SDFS, there is a master server that maintains information about each file such as the holders of each file and latest version of each file, receives request from other machines, and is responsible for deciding which file get stored where. In addition, the above-mentioned information is also implicitly maintained by other three machines so that there always exists at least one potential master. In the case that current master is down, among those three machines that also maintain the file information, the machine with lowest id will be elected as new master. In the case that non-master machine is down, master will detect the failure and let another machine get all the files originally stored on the failed machine, so that there are always four replicas for each file in SDFS. Further-more, there is a separate process on each machine which maintains the current master and writing the current master on disk, so that when a machine failed and rejoins the system, it knows the current master in the system.

1.2 Supported Operations

(1) PUT
The machine *i* that initiates the PUT operation will first contact the master, and master will then return a list of machines (in our system, four machines) on which the new file will be stored. After receiving the list, machine *i* will then send the file to each of machine in the list. Therefore, there are four replicas for each file in our system. Additionally, each PUT operation needs to be ACK-ed by W=3 machines, and only then PUT operation is finished.
(2) GET
The machine *i* that initiates the GET operation will first contact the master. If the file being requested exists in the distributed file system, master will then return a list of machines (in our system, two machines) that hold the requested file. After receiving the machine list, machine *i* will then send "Get Latest Update Time" request to each machine in the list and this request is used to obtain the latest update time of the requested file on each machine. After that, machine *i* will send GET request to the machine that holds the latest requested file, and store the file in local directory. Also, in our system, it is evident that each GET is ACK-ed by R=2 machines.
(3) DELETE
The machine *i* that initiates the DELETE operation will send DELETE request to master, and then master will send DELETE request to machines that hold the file to be deleted. After receiving REPLY from all file holders, master will reply with ACK.
(4) LS
The machine *i* that initiates the LS operation will send request to master, and master replies with list of machines that hold the file.
(5) STORE
Each machine maintains list of all files currently being stored on itself.

(6) GET-VERSIONS
The first part of GET-VERSIONS is the same as GET operation, including obtaining list of file holders and corresponding latest update time. Then machine *i* will send GET-VERSIONS request to the machine that holds the latest requested file, which will transfer all versions of the requested file back to machine *i*.

1.3 Use of MP1 and MP2 in MP3

MP2 is used to maintain membership list on each machine and detect machine failure. In addition, whenever a file is to be put, got, or deleted, the associated machines must be alive, so each operation involves inspection on the membership list. MP1 is used to debug MP3 by grep on files being fetched to local directory, files being uploaded to SDFS, so that we can check whether uploaded files are correctly replicated, or whether files are fetched correctly with specified versions.

2. Measurement

(i) Re-replication Time and Bandwidth
We carried out the measurement with 40MB file when machine failure happens for 5 times. The measured re-replication time are: 192ms, 194ms, 191ms, 185ms, 155ms.
It follows that $T_{avg} = 183.4ms$, $T_{std} = 14.5ms$. And the confidence interval with 95% confidence level and degree of 9 is [163.2521, 203.5479]. Thus, we can conclude that there is a 95% probability that that the re-replication time is within [163.2521, 203.5479].
And we used **iftop** to measure the bandwidth when machine failure happens, the average bandwidth of re-replication is **14.6Kb/s**, and the bandwidth of the machine that is responsible for replicating the file is **51.2Mb/s**.

(ii) Times to Insert, Read, and Update (with 5 trials for each data point)
For 25MB file:
Insert: 424ms, 299ms, 297ms, 312ms, 322ms, $T_{avg} = 330.8ms$, $T_{std} = 47.48ms$.
Update: 207ms, 226ms, 198ms, 252ms, 206ms, $T_{avg} = 217.8ms$, $T_{std} = 19.42ms$.
Read: 196ms, 161ms, 127ms, 62ms, 66ms, $T_{avg} = 122.4ms$, $T_{std} = 52.45ms$.
For 500MB file:
Insert: 3319ms, 2984ms, 2830ms, 3300ms, 3087ms, $T_{avg} = 3104ms$, $T_{std} = 186.77ms$.
Update: 3708ms, 3628ms, 3264ms, 5444ms, 3114ms, $T_{avg} = 3837.6ms$, $T_{std} = 830.95ms$.
Read: 3257ms, 2022ms, 2892ms, 1228ms, 2632ms, $T_{avg} = 2406.2ms$, $T_{std} = 713.67ms$.

(iii) Function of `get-versions` and `num-versions`
We measured GET-VERSIONS command with **25MB** file, where each `num-versions` is measured 5 times. The average time to perform `get-versions` for each `num-versions` are:

Num-versions	1	2	3	4	5
Time	80.4ms	183ms	184.6ms	233ms	272.8ms

The plot of function is shown in Figure 1.

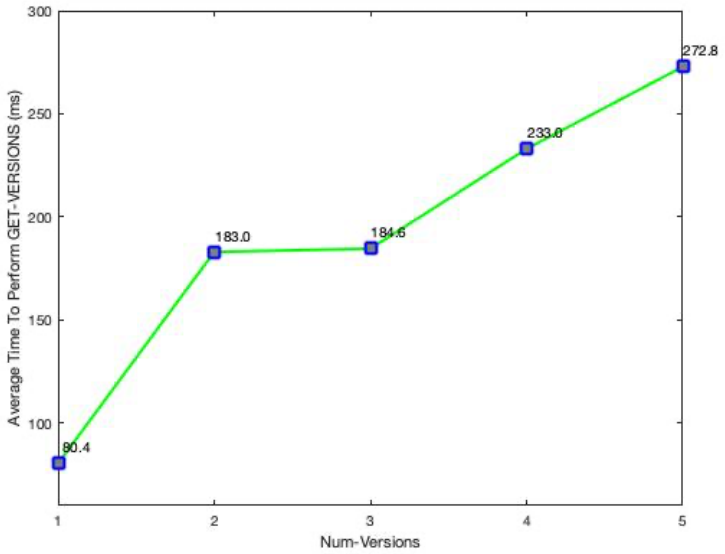

Figure 1 Time to perform get-versions as a function of num-versions.

It can be observed from the measurement that the function between num-versions and time is not linear, and this is because we created multiple threads to get all versions of the file concurrently, thus it would be a lot faster than get all versions one by one in a serial manner.

(iv) Time to Store English Wikipedia Corpus into SDFS

With 4 machines: the measurements are carried out 5 times, and the results are: 15150ms, 12410ms, 11475ms, 13240ms, 12942ms, $T_{avg} = 13043.4ms$, $T_{std} = 1212.25ms$.

With 8 machines: the measurements are carried out 5 times, and the results are: 12304ms, 12149ms, 10920ms, 12076ms, 12254ms, $T_{avg} = 11940.6ms$, $T_{std} = 516.46ms$.

The plot is shown in Figure 2 and Figure 3 as follows.

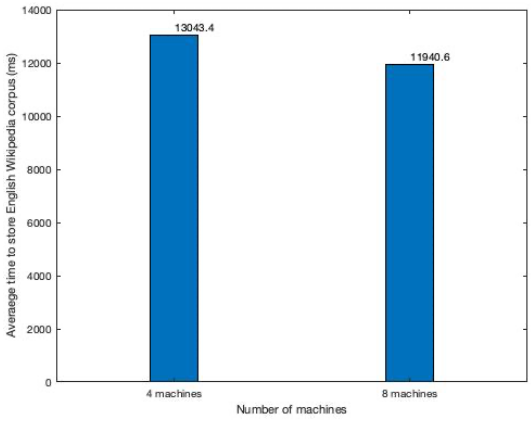

Figure 2 Average time to store Wikipedia

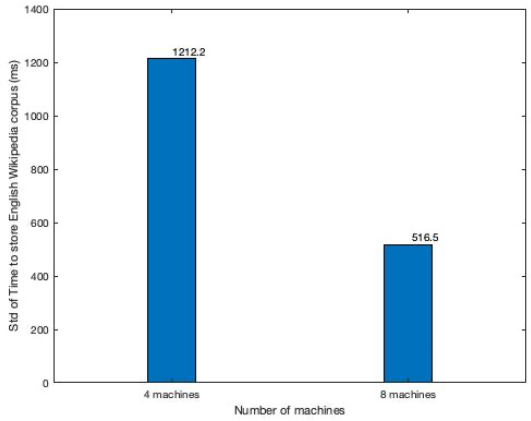

Figure 3 Std of time to store Wikipedia

It can be observed from the figure that the time to store English Wikipedia Corpus into SDFS with 4 machines is almost the same with 8 machines, and this is because in our system, when inserting files into SDFS, there are always four machines are asked to store new file, thus even though there are 8 machines alive, the result is the same as when there are 4 machines alive.