# Machine Programming 3 – Simple Distributed File System

**a) Design**

For this distributed file system, we built our project based on the MP2 membership protocol, and can tolerate 3 simultaneous failures of machines in our clusters. We design a paxos-like replica algorithm so that each time we put a file, we will guarantee that in a certain time period, 4 alive machines will store the replicas of this file. And the system will eventually be consistent.

**Clients** will send commands to a separate process, and clients will support the 6 sdfs commands: put, get,delete, ls, store, and get-versions. It keeps a store dictionary.

**The separate process** always knows who is the master now(by ping-acking the master in each time period), and will forward every command of the client to the current master. When it detects that the master fails,it will initiate a leader(master) election. It will call every machine in the cluster to elect a new master, using bully algorithms.

**The master** handles every command sent from the separate process(originally sent from clients) and decides where to replicate or transfer each sdfs-file. The master knows the following information of the sdfs-system:

- a dictionary maps from file to N machines it is stored in. (ls)
- (We can also map each version of each file to N machines.)
- a dictionary maps from file to its latest version number. (num-version)
- a dictionary maps from machine id to its sdfs-files stored in it. (store)

**Logic of each command**

if put: master send 4 ip address to client. client will receive a master’s instruction and send a replica to N=4 machines(4 ip address), then the client will parallelly send the file to these 4 replica machines. if get/get-version: client will receive a master’s instruction to receive a replica from 1 ip address.This machine will send the file(or merged versions) to client. If delete: client does nothing. Master receives command and deletes the file dictionary.If store: return local store dictionary.If ls: Master send the return dictionary to client.

**Replica algorithm:**

Since we need to guarantee 3 simultaneous failures will not affect our system, we set the number of replicas of each file **N=4**. Write replica count **W=4**. Read replica count **R=1**. So that we satisfy two necessary conditions: 1. $W+R > N$;2. $W > N/2$.

This condition is ($W=N$, $R=1$), which is great for read-heavy workloads.

**b)Past MP Use**

The MP2's membership protocol using in two parts. First, the master process uses the protocol to decide which process should store the file. Once a new file is stored in sdfs, the master will detect the "RUNNING" processes and choose 4 "RUNNING" processes to store the files. Second, once some processes are detected as "LEAVE", the master will choose other "RUNNING" processes to store the files that are stored in the "LEAVE" process.

The MP1 using in checking the files are stored and replicated correctly. Once a new file is stored in sdfs, we will use the MP1 to send a grep command and compare the result with the original file. If the result matches, which means the store is correct. The same idea is used in the failure replica. Once a process failures, the files stored in the failure process would be replicas in other processes. Again, we will use MP1 to send a grep command and compare the result with the original file to ensure the replica is correct.

**c) Measurements**

(i) re-replication time and bandwidth upon a failure (measure for a 40 MB file)

| | test1 | test2 | test3 | test4 | test5 | average | standard deviation |
|---|---|---|---|---|---|---|---|

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| Times | 1.325 | 1.217 | 1.221 | 1.258 | 1.196 | 1.243(s) | 0.05 |
| Bandwidth | 3.773Mbps | 4.108 | 4.095 | 3.974 | 4.18 | 4.026 | 0.159 Mbps |

Table 1. Re-replication time and bandwidth upon a failure

The average time for re-replication of a 40 MB file when failure is 1.243 seconds, and the average bandwidth is 4.026Mbps/seconds. The average time approaches insert files since the re-replica and insert have the same logic.

(ii) times to insert, read, and update, file of size 25 MB, 500 MB (6 total data points), under no failure

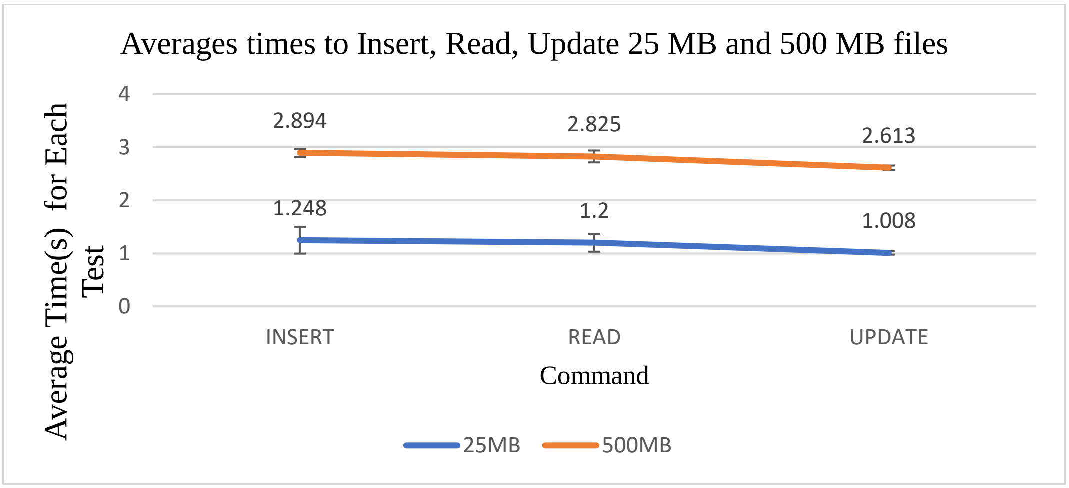

Fig 1. Averages times to Insert, Read, and Update 25 MB and 500 MB files

The average time for inserting 25 MB and 500 MB files is 1.248 and 2.894 seconds, and the standard deviation is 0.507 and 0.152. The average time for reading 25 MB and 500 MB files is 1.2 and 2.8254 seconds, and the standard deviation is 0.339 and 0.224. The average updating 25 MB and 500 MB files time is 1.008 and 2.613 seconds, and the standard deviation is 0.064 and 0.081. The 500 MB file has a higher transmission time than 25 MB when inserting, reading and updating files, and the reason is larger file size needs a higher transmission time. Compared with test 4 "store entire English Wikipedia corpus", file size significantly influences transmission time more than the number of machines if we use thread to send files simultaneously.

(iii) Plot the time to perform get-versions as a function of num-versions

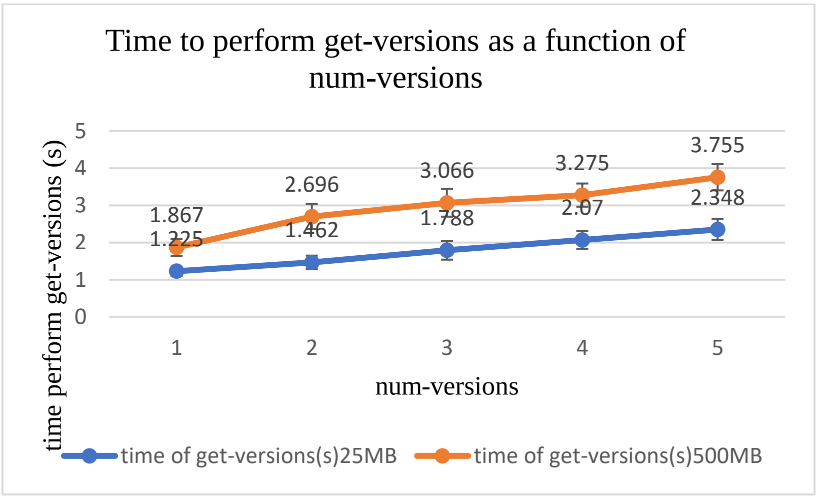

Fig 2. Time to perform get-versions as a function of num-versions

We can see that as the parameter num-version increases, the time to perform get-versions will slowly increase, and slower than a direct ratio. Num-version 1 to 5 only from 1.225 to 2.348. Maybe it's because the size of a file when transferring through the network is not a bottleneck of this function, but other IO or computation processes. Besides, a 500MB file will take more time than a 25MB file when the num-versions are the same, but not 20 times but around 2 times, so it increases much slower than direct ratio.

(iv) time to store the entire English Wikipedia corpus into SDFS with 4 machines and 8 machines

| | test1 | test2 | test3 | test4 | test5 | average | standard deviation |
|---|---|---|---|---|---|---|---|
| 4 machines | 6.561 | 5.799 | 4.897 | 5.382 | 5.694 | 5.666(s) | 0.6106 |
| 8 machines | 6.604 | 5.866 | 5.512 | 6.262 | 6.118 | 6.072(s) | 0.4114 |

Table 2. Store English Wikipedia corpus times in 4 and 8 machines

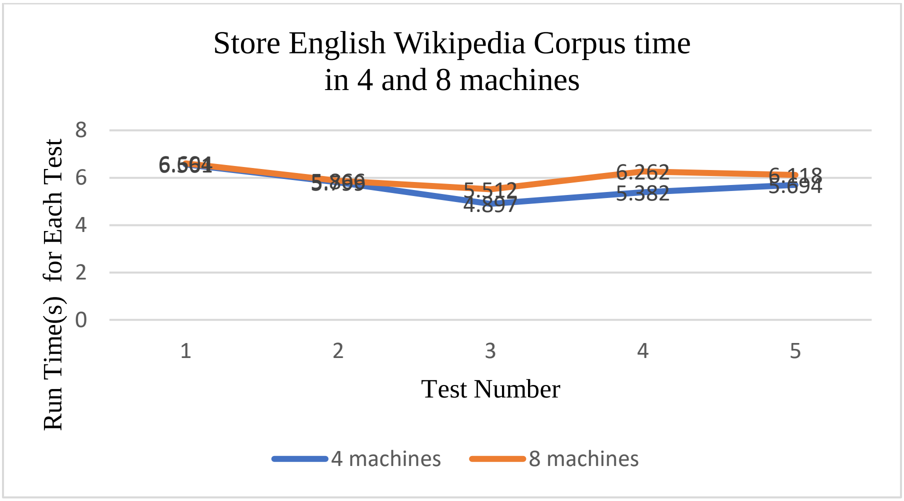

Fig . Store English Wikipedia corpus times in 4 and 8 machines

The results for storing the whole English Wikipedia corpus time in 4 and 8 machines are shown in table and fig 4. The average time for replica corpus in 4 machines is 5.666 seconds and in 8 machines is 6.072 seconds. The standard deviation is 0.616 in 4 machines and 0.4114 in 8 machines. As the result shows, it had no significantly different stores in 4 or 8 machines since we use thread to send the corpus from master to all the replica processes simultaneously. The store times in 8 machines slightly higher than 4 machines probably result from the master execution codes' times.