CS 425 MP 3 Report

Ruipeng Han (ruipeng2), Tomoyoshi Kimura (tkimura4)

Design

For this MP, we decided to use 3 replicas, meaning there are ***four nodes containing the same file***. We also have a Master node, which is fault-tolerant, storing the meta information about the entire filesystem. Master node acts as a worker node as well (i.e store files put on the SFDS). The first process to join would become the Master and the Introducer.

We use **passive replications** for this MP. Basically, each node consults the Master node first, then the Master node will give directions on what nodes to read/put/delete which files. The nodes will hear from the Master node on replicating a file or sending a file. We have set W=4 and R=1 and have ***W + R = 5***. For instance, when one performs a PUT on a node, the redirect the request to Master node, who will hash the filename and return the node the current version number and a set of replicas chosen. The same logic holds for read, get, ls, and delete. Since a master is handling all the requests, the master acts as a sequencer that ensures every operation is in total ordering.

During file transfer, each file is not sharded, and each node could store the different versions for the file. We have built our code on the previous MP. However, we have modified it such that any node could be the master/introducer, while previously we only fixed one machine to be the master/introducer in the previous MP. To do this, we have implemented a simple election algorithm in which each node chooses the highest member in the membership list to be the leader, taking full advantage of the full membership list and the reliable failure detector. When the master detects/receives a node failure (via our failure detector), the master pulls up the failed process' metadata and **re-assigns each of its files to a different replica to ensure the number of replicas is still 4**. We also implemented an election algorithm so a new master/introducer would be elected after the previous one failed. The election is implemented using the Bully algorithm with multicast and selecting the process with the highest id as the new leader.

Past MP Use

MP 1 Distributed Logger

- MP 1 is extremely helpful for debugging. Our programs set up a log file in the beginning of the process, and every action, failure, and error, is logged to the log file. Therefore, we often use this distributed logger to get all the log files across the virtual machines for debugging.

MP 2 Failure Detector

- Our failure detector has laid the foundation for this MP. We use the full membership list to directly send files to other nodes, and use the failure detector to ensure that upon any node failure, the files stored on the node is quickly replicated to another node.

Measurements

Re-replication time and bandwidth upon a failure - 40MB file

The bandwidth is expected, since only one node would send the file to another node. The time is consistent and fast. It is fast, since we only put one file once.

Trials	Trial 1	Trial 2	Trial 3	Trial 4	Trial 5	Mean	std
Time (s)	0.187287	0.196365	0.200967	0.199635	0.186867	0.1942242	0.0067375
Bandwidth (MB)	41.2	41.2	41.2	41.2	41.2	41.2	0

Times to insert, read, and update files

Insert

We have implemented concurrent uploads to different replicas. Uploading a 500 MB file takes way longer than the 25 MB file. It is surprising that a 500 MB file has 7.7 seconds, but the mean time to upload a file is not in a linear relationship with the file size. The bigger the size, the longer it will take.

Trials	Trial 1 (s)	Trial 2 (s)	Trial 3 (s)	Trial 4 (s)	Trial 5 (s)	Mean (s)	Std
25 MB	0.229641	0.336866	0.255802	0.252866	0.205738	0.2561826	0.049421
500 MB	7.733678	32.75468	26.94962	26.17284	54.25091	29.572345	16.69222

Read

Read is similar to the insert. Uploading a 500MB file is about four times less than insert, which sounds reasonable since we are only requesting for the file once if successful.

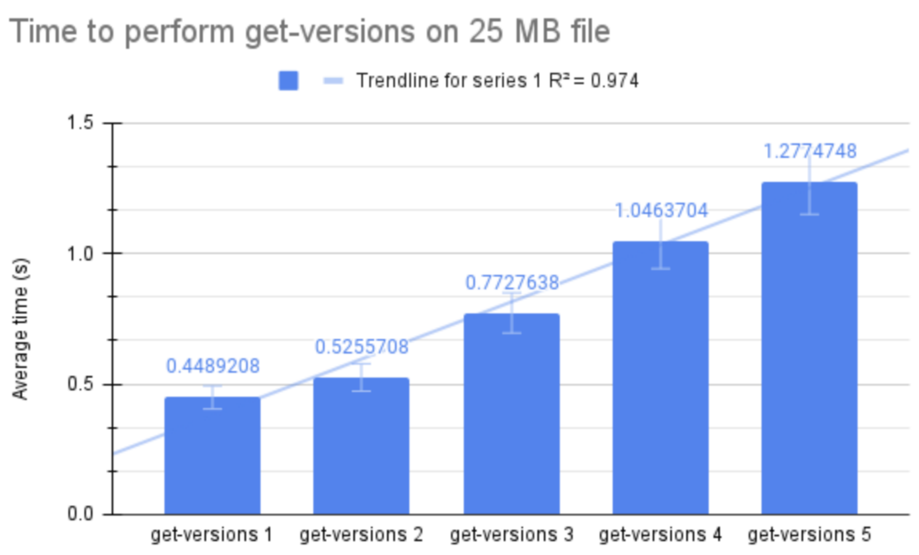
Trials	Trial 1 (s)	Trial 2 (s)	Trial 3 (s)	Trial 4 (s)	Trial 5 (s)	Mean (s)	Std
25 MB	0.144915	0.160974	0.150987	0.146713	0.142302	0.149178	0.007313
500 MB	4.12237	5.314137	4.439959	6.647050	11.192473	6.335124	2.886034

Update

The Update is similar to the Insert, since they have similar behaviors.

Trials	Trial 1 (s)	Trial 2 (s)	Trial 3 (s)	Trial 4 (s)	Trial 5 (s)	Mean (s)	Std
25 MB	0.221948	0.293839	0.2628	0.256683	0.22248	0.2515036	0.0303156
500 MB	10.16438	40.29805	34.79688	22.21761	34.36516	28.368421	12.132824

Time to perform `get-versions` with 25 MB file



The trend presented is a linear relationship between the average time and the number of versions requested. This is reasonable since the number of requests increases as the number of versions increases. The standard deviation reflected by the error bar also indicates that the variance increases as the number of requests increases. With a higher bandwidth, there are more uncertainties and could result in higher error.

Time to put Wikipedia English Text

Trials	Trial 1 (s)	Trial 2 (s)	Trial 3 (s)	Trial 4 (s)	Trial 5 (s)	Mean (s)	Std
4 Machine	267.8904	323.2312	293.2114	224.0241	343.9316	290.45968	47.07415
8 Machine	143.7577	248.5961	310.8952	291.5613	340.8692	267.13589	76.67135

Wikipedia English Text has a relatively high time. The higher bandwidth has resulted in a high latency in our file uploading process. However, the number of machines has had a small effect in the meantime. This is because we have 4 replicas for both 4 machines and 8 machines.