CS 425 MP 3 Report

Group 55: Yuchen Liang (yliang35) and Xilun Jin (xjin12)

I. Introduction

In this MP, we implemented a distributed file system similar to HDFS. Each node has a part of the entire dataset, and each data piece is replicated at 4 machines (discussed below). Note that we do not use a master in the design.

II. Algorithms

1. *Insert and update*

The system puts a file onto the machines corresponding to the hashed key. On an insert or update command, the file is written to the first 4 machines with ID larger or equal to the hashed key. The past version is kept for later retrieval.

Every time we update all 4 replicas. If new node joins, the system checks whether the send-to list, kept by every node for each file it has, needs to change. We choose to replicate at 4 replicas because there are at most 3 simultaneous failures in the system. In other words, W = 4 (ALL).

2. *Read*

On a read command, the file is fetched from any of the 4 machines, where we choose the first machine for convenience. It always returns the most up-to-date file since the first 4 replicas always contain the latest version. In other words, R = 1.

3. *Re-replication*

On a machine failure, the file is re-replicated at the first node after the old replica node group (w.r.t node ID). This can be accomplished by any of the four machines, and we choose the machine with the least ID to accomplish the task. The send-to list mentioned above is also updated on each relevant machine.

III. Use of MP1

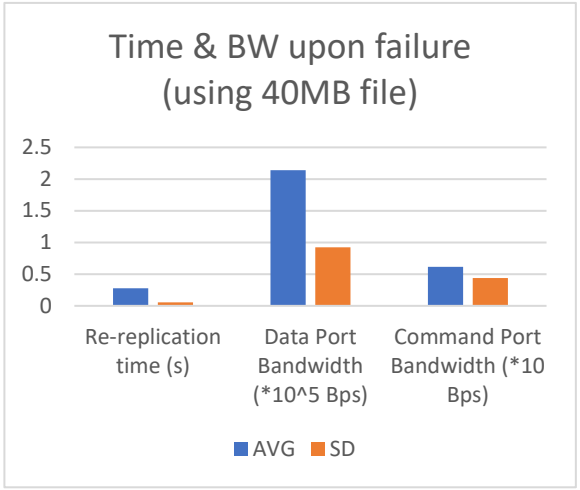
The distributed log system is of great help in MP3. On each put, get and update, we create a log at the current machine. We can use MP1 to grep them together and compare the sender and receiver processes.

IV. Data Analysis

Note on units: Time in second & Bandwidth in Bps

1. *Re-replication time and bandwidth upon failure (~40MB file size)*

	Re-replication time	Data Port Bandwidth	Command Port Bandwidth
1	0.308	261597.82	0.79
2	0.2029	277481.95	5.97
3	0.3342	84673.57	12.97
4	0.239	149135.17	5.26
5	0.299	296493.52	5.77
Avg	0.2766	213876.406	6.152
Std	0.0540	92240.700	4.365

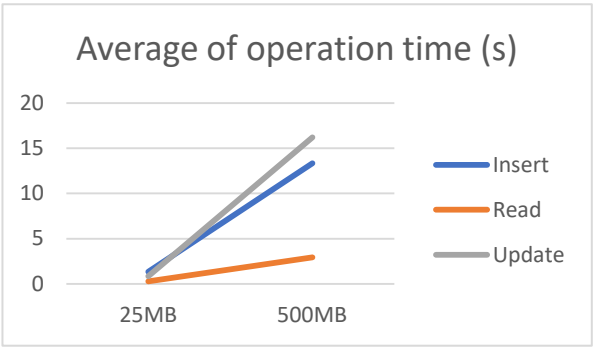


The re-replication time is all within a single membership update time (0.5s). The data port is using most of the bandwidth for file transfer, and the command port is also communicating information, but using a much less bandwidth.

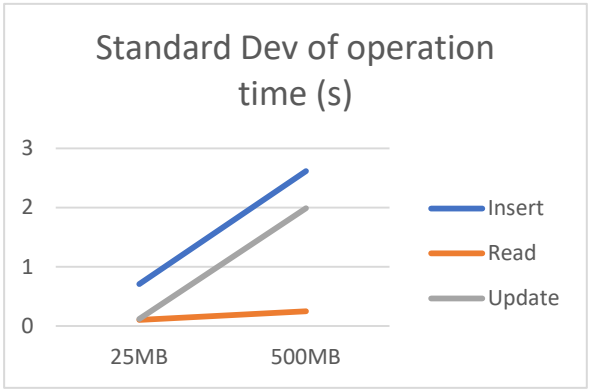
2. *Times to insert, read, and update (no failure)*

25 MB file	Insert	Read	Update

1	1.3077	0.2865	0.9186
2	0.8281	0.2213	0.8958
3	0.6831	0.1986	0.6938
4	2.4827	0.4320	0.7815
5	1.4639	0.1744	0.9877
Avg	1.353	0.2626	0.855
Std	0.710	0.103	0.117



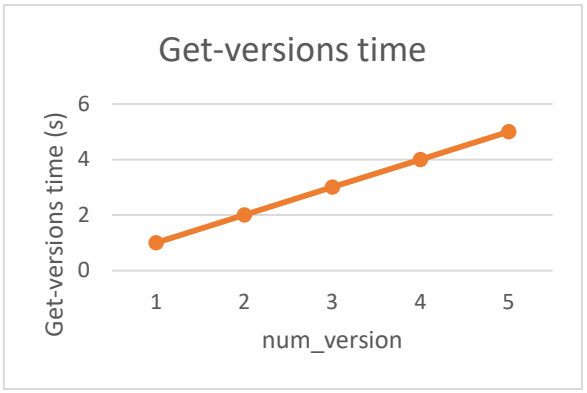
500 MB file	Insert	Read	Update
1	12.49	3.31	16.28
2	13.23	2.81	19.54
3	11.43	2.76	14.90
4	11.69	3.04	14.49
5	17.84	2.71	15.92
Avg	13.34	2.93	16.23
Std	2.615	0.249	1.991



Both the average and standard deviation of operation time increases when the file size increases. Insert and update operation take similar time because every time we write to all replicas (W=ALL). The read operation is taking a much less time since we only read from one replica (R=1).

3. Get-versions command (using 25 MB)

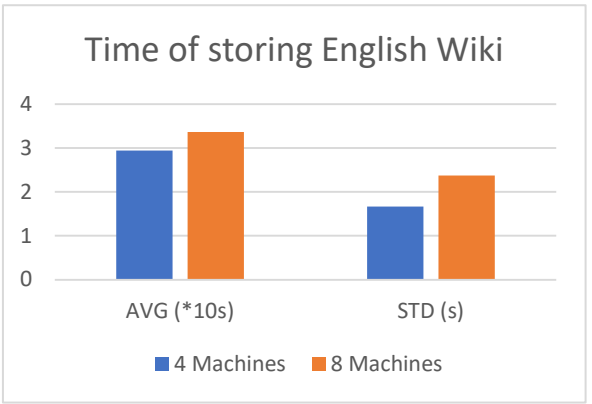
Observations	num_version	Get-versions time
1	1	0.2057
2	2	0.4361
3	3	0.5443
4	4	0.8080
5	5	1.1029



The time for get-versions is almost linear with num_version. This is because for every extra version we grab one more file.

4. Times to store the entire English Wiki

	4 Machines	8 Machines
1	29.68	32.66
2	31.51	35.16
3	27.76	29.98
4	30.46	35.98
5	27.75	34.31
Avg	29.432	33.618
Std	1.6629101	2.376366975



The storing time of the entire English Wiki using 4 and 8 machines does not change much. This is because the system only writes to 4 replicas, so there is not much difference if increased from 4 to 8.