CS425 MP3: Simple Distributed File System Report

Rohan Jyoti (jyoti1), Yiwei Yang(yyang117)

General Algorithm:

There are 3 types of machines: mDNS, mServer, and mClient. mDNS is a fixed machine (assumed to not fail) that stores the current leader's IP and Port. mClient will resolve the master machine through mDNS. Each mServer instance is associated with an mServerType (either MASTER or SERVANT), and will execute accordingly. mClient can only communicate with mDNS and the master mServer.

Upon a PUT, the master server will split the file and distribute evenly within the SDFS system with a replication factor of 2. For example, if mClient uploads a 1GB file and there are 4 servers, each server will have 1 original and 1 replica each of size 250MB. Thus, our system employs an **even-distribution load-balancing scheme.** The original and replica chunks stored on each machine are not the same chunk, however (the primary and secondary replicas are never stored on same machine). For example, if there are 4 machines, a file will be split into 4 chunks and for each mServer instance *i,* it will have chunk *i* of type original and chunk *i+1* of type replica. The distribution scheme is explained in more details within the code.

Upon a GET, the master server will send any relevant chunks it may have, and then forward the request to the replication servers who will then accordingly connect to mClient and **push** the download.

The failure detection protocol employed is the **Ring Heartbeat Protocol**; when an mServer instance detects predecessor failure, it TCP multicasts the failure. The leader election protocol is **custom**: It is guaranteed in our system that the membership list for each mServer instance is consistent; therefore, it is logically sound for each mServer instance to simply elect the first non-faulty member in their membership list. The type of replication strategy employed is **active replication**; as long as there at least 2 servers, it is guaranteed there will be two replicas.

Using MP1 would theoretically be helpful in debugging MP3; however, since we created a custom mprintf (re-implementation of printf) that allowed us to write both to screen and log file, its use was extremely limited.

Measurements:

i. Re-replication time and bandwidth upon a failure (1GB file):
   Re-replication time = number of replicas (2) * time it takes to transfer and write chunk from mServer to mServer ≅ 9.2 seconds
   Bandwidth upon failure is equivalent to size of chunks contained on the failed mServer → 250 * 2 = 500 MB

ii. Time Between Master Failure and Leader Elected
   Let variable ntwrk_latency represent the time it takes for a simple message to travel from one mServer instance to another. As discussed earlier in the General Algorithm section, our custom leader election allows for an upper time bound of (failure_detection_time + ntwrk_latency).

iii. Read/Write 1MB, 10MB, 100MB, 1GB → Tables and Graphs below

iv. Wikipedia Corpus PUT: 122.7557702 Seconds

Table 1: Write (aka PUT)

Testfile	Trial	mClient PUT	Master mServer PUT	Servant mServer PUT	Total/System PUT		AVERAGE	STDEV
testfile1MB	Trial 1	0.002812	0.00476	0.0080427	0.0156147		0.01424454	0.001340859
	Trial 2	0.003018	0.003902	0.006736	0.013656			
	Trial 3	0.001388	0.004063	0.010315	0.015766			
	Trial 4	0.001774	0.003667	0.007674	0.013115			
	Trial 5	0.001765	0.003773	0.007533	0.013071			
testfile10MB	Trial 1	0.023063	0.035106	0.066476	0.124645		0.1015658	0.014261339
	Trial 2	0.015198	0.028689	0.052268	0.096155			
	Trial 3	0.015238	0.028556	0.043069	0.086863			
	Trial 4	0.016042	0.02746	0.060521	0.104023			
	Trial 5	0.016171	0.027527	0.052445	0.096143			

CS425 MP3: Simple Distributed File System Report

Rohan Jyoti (jyoti1), Yiwei Yang(yyang117)

testfile100MB	Trial 1	0.189124	0.371491	0.56943	1.130045		1.0735724	0.040349392
	Trial 2	0.136523	0.340994	0.616084	1.093601			
	Trial 3	0.129555	0.330522	0.562921	1.022998			
	Trial 4	0.130769	0.329924	0.603666	1.064359			
	Trial 5	0.130418	0.323096	0.603345	1.056859			
testfile1GB	Trial 1	2.83686	5.572365	9.24126	17.650485		17.5365386	0.557910235
	Trial 2	2.54789	5.987083	9.16573	17.700703			
	Trial 3	2.89768	5.763524	8.75632	17.417524			
	Trial 4	2.53265	5.556989	8.59979	16.689429			
	Trial 5	2.98009	5.878982	9.36548	18.224552			

Table 2: Read (aka GET)

Testfile	Trial	System GET		AVERAGE	STDEV
testfile1MB	Trial 1	0.013151		0.0133384	0.000414915
	Trial 2	0.013655			
	Trial 3	0.012938			
	Trial 4	0.013052			
	Trial 5	0.013896			
testfile10MB	Trial 1	0.106148		0.1182562	0.010801131
	Trial 2	0.123598			
	Trial 3	0.124365			
	Trial 4	0.129854			
	Trial 5	0.107316			
testfile100MB	Trial 1	2.766047		2.7928132	0.035879466
	Trial 2	2.837556			
	Trial 3	2.776873			
	Trial 4	2.824771			
	Trial 5	2.758819			
testfile1GB	Trial 1	33.378858		33.7042124	0.644378996
	Trial 2	34.547876			
	Trial 3	34.236113			
	Trial 4	33.112456			
	Trial 5	33.245759			

Note: Overall, the graphs for these data points are useless due to the huge variance between (1MB, 1MB, 100MB) and 1GB. Therefore, just for the sake of showing trends (the standard deviation bars are indeed there, just too small to see):