CS425 MP3 Report

Taipeng Liu(taipeng2) & Xiang Li(xiangl14)

1. Design

Algorithm: Our TCP-based SDFS is a simplified distributed file system which implements remote procedure calls (RPCs) from client to server and server to server. We use Go and its package "net/rpc" for this assignment. Each machine acts as both a client to access local file and a "datanode" server which stores SDFS file replicas. As we implement a master-slave model, a cluster will have exactly one machine running "namenode" server that processes read/write commands. On sending put, get or delete requests, client will firstly get a list of datanodes from the namenode, and then executes corresponding operation by connecting to those datanodes. File is not shared, i.e. it is stored entirely. However, an entire file is sent by a buffer with 10MB size. For example, a 40MB file will be sent 4 times to one datanode.

Replication Strategy: Namenode deals with placement of each file replica and re-replication when failures occur. We apply passive replication with 4 replicas for each file to tolerate at most 3 simultaneous failures. Namenode maintains a file map to store SDFS file metadata including last modified time and a list of nodes who store the file, and a node map which records what SDFS files are stored in each datanode. The correctness of these two maps is crucial for our SDFS, so each time client inserts, updates, deletes or failure happens, namenode updates those maps and decides whether re-replication is needed.

Quorum: For 4 replicas we use a quorum with W = 4 and R = 1 which achieves fast read and assures consistency. Since client runs each operation in many Goroutines, we use mutex lock when a Goroutine finishes and add 1 to the response count. Once the count equals to W in write or R in read, client returns.

Usage of MP2&MP1: Our MP3's implementation highly relies on MP2's membershiplist to check failed nodeID then ensures fast and accurate re-replication, and we assume a reliable fail detection algorithm. MP1 is useful for querying log files remotely from one client. As before, we create "MP3_[# of VM].log" file on each VM for reference. We print log with keyword of corresponding function name or which will helps us pinpoint quickly, i.e. "grep -E putfile" to get related logs of function putfile.

2. Measurement

a. Analysis of re-replication time and bandwidth for a 40MB file

Table1.

	Re-replicate time (ms)	Bandwidth (MBps)
Average	121.5475394	329.11
Standard Deviation	15.34564606	24.69

We launch 10 VMs and insert a 40MB file to the cluster. Then, we sequentially fail up to 5 nodes to have 6 datapoints in total. Re-replication is done by function "checkReplica". The time is measured from when checkReplica detects there is not enough replicas for a file to when the function returns. Therefore, the time doesn't contain time for failure detection and shows re-replication time cost for only one file. If the failed node stored several files, then the function would be called more than one time and measures re-replication time for each file.

b. Analysis of insert, read, and update, file of size 25MB, 500MB, under no failure

Our experiment has 6 data points in total, each data point is measured by 5 trials.

Table2.

	25MB		500MB	
	Avg. (ms)	S.D. (ms)	Avg. (s)	S.D. (s)
Insert	229.829735	25.08547587	3.195101563	0.383215109
Read	160.9480812	28.08285521	2.182874151	0.575920602
Update	192.9468726	23.0926907	3.22327411	0.445770547

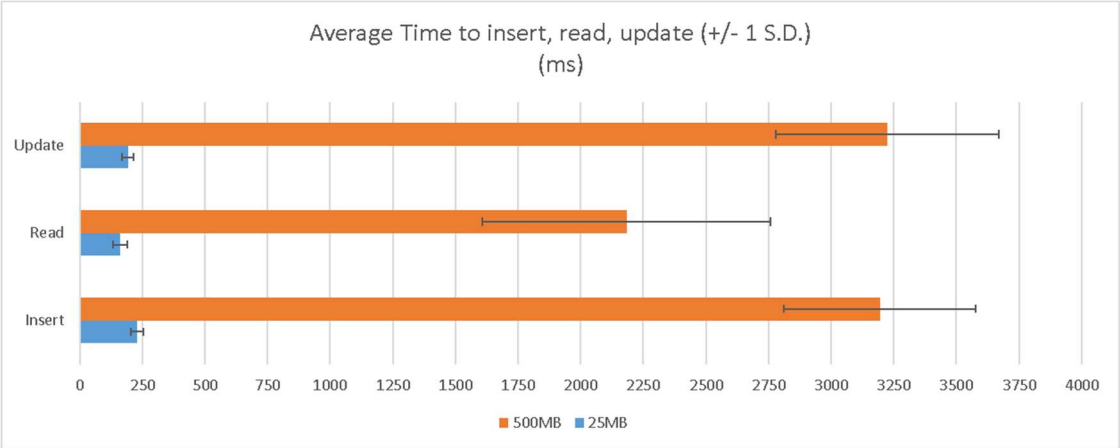

Fig1. Avg. Time to insert, read, update (+/- 1 S.D.)

While average time for operating a 500MB file is about 15 times larger than 25MB file, the standard deviation for a 500MB file is also greater than the other. This may result from the internet delay, which can probably occur more times in write/read of a larger file. For each file, different file operations have the same time cost trend: Update ≈ Insert > Read. Since each file is inserted or updated entirely, these two operations are much the same that requires W = 4 before return. However, read only wait for R = 1 datanode response, time cost must be less.

c. Analysis of store the entire English Wikipedia corpus with 4 and 8 machines

Table3.

	4 VM	8 VM
Average (s)	69.911	42.954
Standard Deviation (s)	25.893	6.085

Typically, time of storing the entire English Wikipedia corpus with 8 machines is less than storing with 4 machines, because on average each machine stores less files.