CS2510 Course Project: Fault Tolerant Distributed Storage System

Authors: meiqi.guo@pitt.edu; yud42@pitt.edu
Reviewers: lol16@pitt.edu

Introduction

In this project we implemented a distributed file storage system which consists of three components: Directory server, Storage Nodes and Clients. The file system is able to handle file adding as well as reading operation in a fault tolerant principle based on our design. We also programmed a family of test programs which can be used to test simultaneous multi-requests as well as multiple fault-tolerant scenarios.The entire project is implemented by Python, and can be simply run on the element servers with command line parameters. Based on our program, we evaluated multiple fault-existing situations such as directory server down, multiple storage nodes down and both.

Frameworks

In this section we briefly introduces our design as well as program design. As we stated in the design document, our project is programmed in three parts: FileSystem.py, utils.py and test scripts which contains run_directory_server.py, run_storage_node.py and run_client.py. The detailed introduction of each part are as follows:

- FileSystem.py: In this file we defined all the basic classes for the project, includes classes for directory server role as class DirectoryServer, storage server role as class StorageServer and client role as class Client. Additionally, we specified predefined constants such as: publicly known directory server address, experiments needed addresses for storage nodes. The servers are able to handle simultaneous requests by multi-threadings. More overly, the statistics of each role will be recorded.
- utils.py: In this file we defined several functions used for our programming.
- run_directory_server.py: This program is used to run directory server role on predefined element, the program will launch two directory server (one as primary directory server and another as backup directory server). Also the program provides a command line parameter to specify when will the primary directory server down. Details seen in manual. The program will show detailed log about directory server running history will be shown, and the communication statistics (messages transferred, bytes transferred) will be shown once it is shut down.
- run_storage_node.py: This program is used to run storage nodes on predefined element, the program will launch three storage nodes (one as primary storage, rest as backup storage nodes). More storage nodes will be launched by communication from directory server once the primary storage is down and detected by either a client or directory server itself. More details about input parameters in manual.
- run_client.py: This program is used to run client with required parameters as configs for experiments, including file size (M), request number (M) and frequency of requests (F). It is worth mentioning that we compacted 4-step requests together for a comprehensive examination: a connecting request which used to get a storage node,

an add file request which used to randomly add a file to the file system, a get list requests which used to get the file list from the file system and finally a read file request to download/read a random file from file system. The final statistics include average response time, number of messages transferred, number of bytes transferred will be shown in the log in the end.

Experiment Setup

In this section we described the experiments setup for our analyses. According to the requirements, we evaluated our design based on **total messages transferred**, **total bytes transferred** and **average response time** on client side. The directory servers, all storage nodes and clients are run with different addresses. To make the experiment as close as possible to real situations, we run the directory servers, storage servers and clients on different elements. To modify the configs for addresses of each role family, simply change constants in FileSystem,py.

The test cases as well as their results are shown in the following section.

Results

The following table shows our experiment configurations and corresponding results.

Fault number in directory server	Fault number in storage server	M	N	F	Message	Byte	Average Response time
0	0	10	5	0.1	77	9906	0.009361
1 (T=15)	0	10	5	0.1	97	13933	0.014904
1 (T=15)	1 (T=15)	10	5	0.1	134	20913	0.017445
0	2 (T=15)	10	5	0.1	135	19283	0.030132
0	0	10	10	0.1	157	22792	0.005894
1 (T=2)	0	10	10	0.1	184	24363	0.011127
1 (T=2)	1 (T=2)	10	10	0.1	211	28351	0.019037
0	2 (T=2)	10	10	0.1	195	28302	0.022052

Analysis

Based on our experiments we have the following observations:

- With faults happened in directory servers and storage nodes, the messages and response time are obviously increased due to additional messages needed to launch new servers/nodes and following synchronization steps.

- Among all cases, two storage nodes down requires most delays since the synchronization steps of storage nodes need most time among recovery events. Also we can see that more storage nodes down higher response time. ((1 directory down)< (1 directory down + 1 storage down) < (2 storage down))

In conclusion, our design successfully met the distributed file system functionalities and shows fundamental fault tolerance in directory servers (at most 1 given 1 backup) and storage nodes (at most 2 given 2 backups).