Fault Tolerant Distributed User Space File System
Design, Tests

Shrivinayak Bhat | EEL 5737

I. Problem Statement

User Space file system is a software interface which enables the users to create their own file systems without modifying the kernel code. A computer system must be modular, fault tolerant & robust. This project is an implementation of a remote file-system which is suitable for a cloud centric computer system. From a file system perspective, the project offers the following features.

1) Modularity – Modularity in a computer system makes the structure simple and helps in containing errors. In this implementation, the data from a file is split of blocks of predefined size and stored. This division of blocks helps is easy transfer of data and also contain errors from propagating. The block size is variable and can be calibrated with respect to performance.

2) Redundancy – The blocks are replicated across data servers in a round robin fashion which enables fault tolerance. This kind of a setup also reduces the load on individual servers

3) Error Correction – Corrupted blocks are detected using the checksum and such blocks are replaced by obtaining data from replica blocks of adjacent data servers

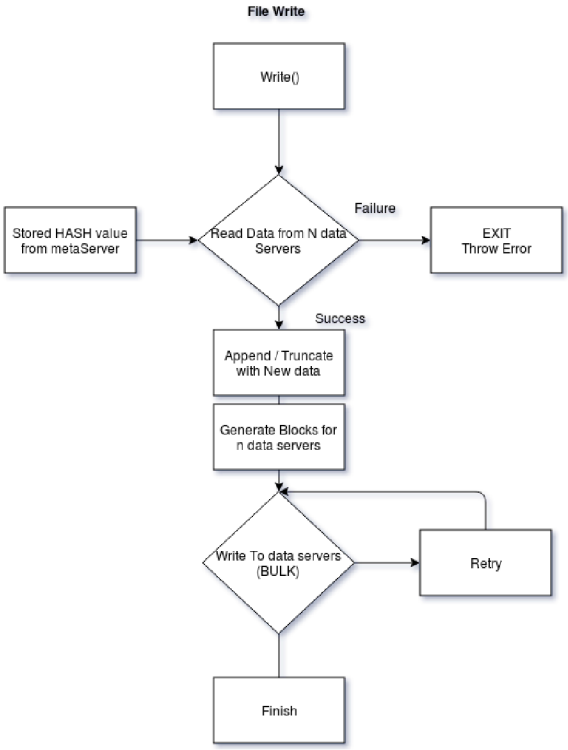
4) Portability – The data servers & meta server are portable. They can be easily moved to any computer on the Internet and can function remotely.

5) Scalability – Scaling is an important factor when considering a distributed system. This design is easily scalable and offers linear increase in performance with addition of data servers

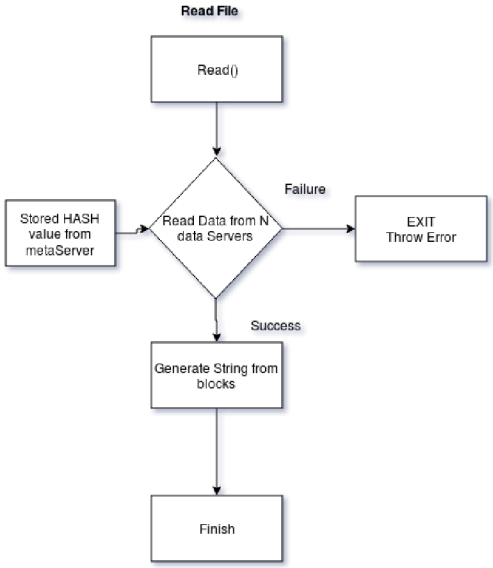
II. Design

The Flow chart for some of the important file system operations is as follows.

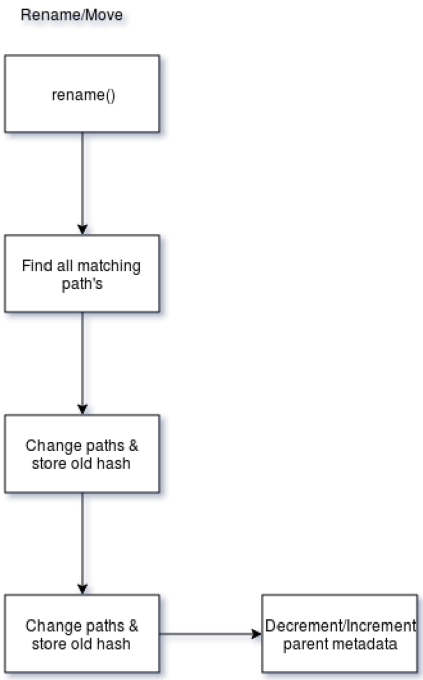
1) Write Operation



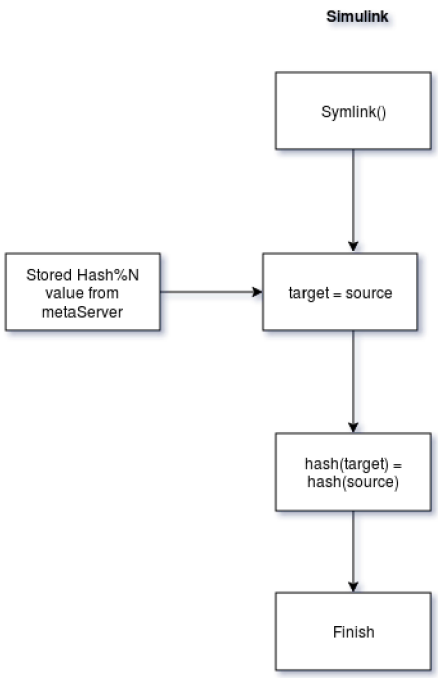
2) Read Operation



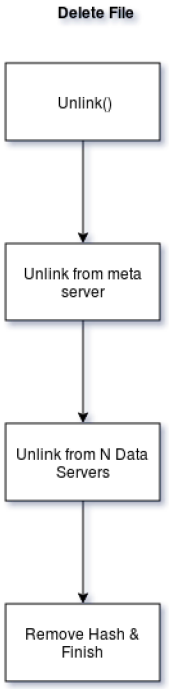
3) Move operation



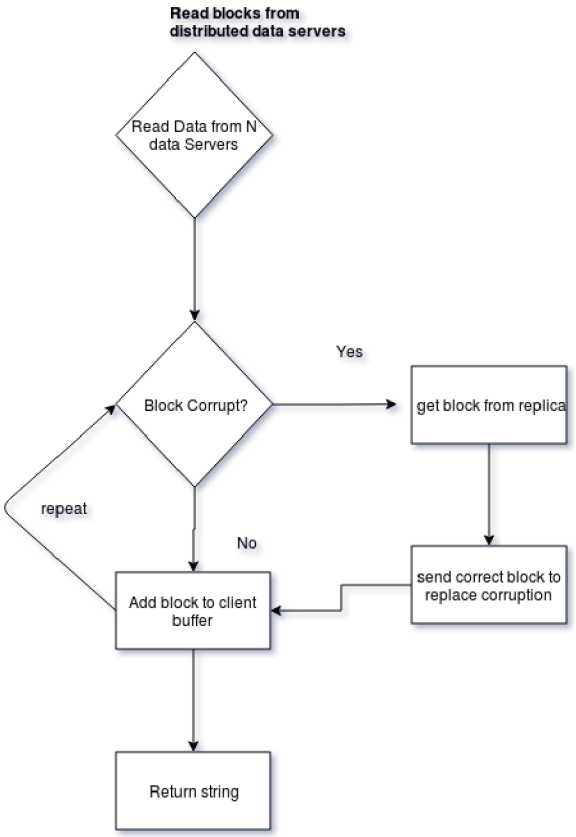
4) Soft-links



5) File Deletion



6) File read from data servers



III. Implementation

Data Distribution:

- Data is distributed across n servers in a round robin fasion starting from an offset calculated using the hash value of the file path
- Replicas of these blocks are stored in adjacent servers in the same round robin fashion
- Each of these blocks are accessible using XMLRPC Calls.

Replication:

- Data blocks are replicated up-to a predefined replication factor. In this way redundancy is achieved

Checksum:

- Each of the blocks has a 32byte checksum associated with it which helps in handling corruption.
- If block size = 8, block size with checksum is 40.
- When a client detects a data corruption in block, it retrieves data from replica block and send it back to the data server for replacement. In this way, corrupted data is contained and immediately corrected.

Write Blocking:

- When any of the data server is down, all write calls from the clients are block and client tries indefinitely until the server is back online
- Reads work fine if any one of the servers containing the block is online.

Persistent Storage:

- All block writes are immediately written to disk under the file name ‘data store’ and in case of a server restart, the data blocks are loaded from the file hence avoiding the problems of volatile memory

Data Recovery:

- In case of server failure with loss of persistent storage, data blocks are retrieved from the replica blocks stored on other servers at the beginning of server start.

IV. Tests

1) File System Functionality Test

Test Case No.	Test	Description	Result
1	No. of Hardlinks for /	Number of Hardlinks should be 2	Passed
2	Write Data into file	Data written using echo/vim	Passed
3	Create Folder Hierarchy	Folder1 > Folder > 2 > Folder3	Passed
4	Check hardlinks for folders	Should be 3 for folder 1 & 2, 2 for folder3	Passed
5	Move files into folder	Move file.txt into folder3 & read data	Passed
6	Soft Links	Creat link to file.txt inside folder3 & read data	Passed
7	Delete File	Remove file.txt	Passed
8	Delete Folder	Remove folder1 & its contents	Passed
9	RMDIR	Folder not empty error if not empty else delete	Passed
10	Read data using symlink	Should read data accurately	Passed
11	Truncate / Append the file	Should modify data accordingly without errors	Passed
12	chown	Change ownership from root to shrivinayak	Passed
13	Data Modification using vim/nano	Should work fine with no erros	Passed
14	Rename Folders with data	All contents should be changed appropriately	Passed
15	Grep search	grep <string> <file>	Passed

2) Distributed System

Test Case No.	Test	Description	Result
16	Checking distribution of data blocks based on hash and replication factor	Blocks should be distributed in a round robin fashion as described in project guidelines	Passed
17	Checking even distribution of blocks among n data servers	Blocks should distributed more widely when no of servers is increased	Passed
18	Load should be distributed & reduced	Read requests make separate calls for each blocks while write happens in a bulk push fashion	Passed
19	Handling read & write operations when hash value changes due to move	When file is renamed, hash changed. Previous hash value is saved and reused to enable no movement of blocks upon renaming	Passed
20	Read data via softlink	File should be readable via soft link. Hash value needs to be handled correctly	Passed

3) Fault Tolerant & Corruption Handling

Test Case No.	Test	Description	Result
21	Corrupted data block recovery on dataserver	Corrupted data blocks are identified on the client, and correct data is sent back to the server for correction	Passed

22	Persistent Storage - shelve	Data blocks should be written to disk immediately when received by the dataserver	Passed
23	Server Restart Recovery	Server should re-populate in memory from persistant storage	Passed
24	Server Restart Recovery -2	Server should recover data from other data servers when restarted on failure & data store is corrupt	Passed
25	Data write operation blocking	Write operation of data from the client should be blocks when any of the data server is down & retired continuously	Passed
26	Data read while failure	Data from the servers should be readable when no 2 adjacent servers have failed	Passed
27	Corruption Simulation	Corrupt one of the data blocks via corrupt.py and see if it recovers when the file is read	Passed

V. Conclusion

The distributed fault tolerant file system has been tested for all end cases. All the features tested above have been verified.