# Fault-Tolerant Remotely Distrubuted File-System Using Upload/Download Model

Sharma, Siddharth
Electrical and Computer Engineering
University of Florida
Gainesville, USA
siddharthsharma@ufl.edu

Tedia, Suvrat
Electrical and Computer Engineering
University of Florida
Gainesville, USA
suvrat17@ufl.edu

***Abstract*—This paper describes the implementation of fault-tolerant remotely distributed file system using FUSE. The system aims to provide resilience towards data corruption and server crashes using data replication on multiple data servers. Utilizing minimalistic-server and smart client framework, the system is optimized to distribute the load across multiple data servers while keeping a track of them in a single meta-data server and data in multiple data servers (at most 5). With the assumption that the meta-server is robust, data replication is used only for data servers where, each of the file data replica server stores a copy of its adjacent server file block. The system can recover from data corruption using the replica copies residing in replica servers. With addition to the capability of recovering (data loss when system crashes) and restoring from a system crash, the model provides non-blocking reads and blocking writes to keep the file system consistent.**

***Keywords—File System, FUSE, Client-Server Architecture, Minimalistic Server, Data Corruption, Fault-Tolerant***

## I. INTRODUCTION

Fault tolerance is the property that enables a system to continue operating properly in the event of the failure of (or one or more faults within) some of its components. A fault-tolerant design enables a system to continue its intended operation, possibly at a reduced level, rather than failing completely, when some part of the system fails. [1]

The ability of a system to rebound from faults and still perform consistently is called fault tolerance [2][3]. The most common approach to make a system fault tolerant is *redundancy* [2]. We achieve redundancy by taking backups where one of hardware systems becomes faulty. The system still provides the required functionality using the redundant hardware.

Remote distributed file system is a file system architecture that enables client's system to remotely operate on file data while providing good access transparency. This system framework provides multiple advantages over a single file server system. Single file server can be a bottleneck [4] in terms of scaling, location transparency i.e. client doesn't keep a track of location of the file; allowing for easy relocation of the file without the need of many changes, provides performance transparency; works efficiency regardless of load and location.

A typical naming scheme like in Unix, uses a hierarchical naming scheme i.e. directories contain other directories. This arrangement of the file system can be approached using two methodologies (Fig 1). Upload/Download model where the client downloads the file, works on it and writes it back. Remote Access model, where the file exist only on the server and client sends commands only to get the work done.

One of the most common approach to make a system scalable is to de-cluster files. De-clustering files means to divide the single large file data into multiple blocks. Later, different blocks can be stored on different servers per a pre-defined rule (round-robin in our case). This not only allows for a proper load balancing, on a scalable network, it will also

reduce the overall network traffic caused due to access to distributed system

We have tested the filesystem for the cases of failure when a server goes down or when the data on a server gets corrupted. The client cannot control the failure if adjacent servers get corrupted. In cases when non-adjacent servers go down, the client identifies the situation and returns the data from the replica data server. On a write request during a server crash, the system operation is blocked to avoid inconsistency in data. Situation where the data gets corrupted on non-adjacent servers, the servers identifies the corrupted data and rewrite their copies with correct data from replica data servers. Also, it provides a sanitation check for the replica servers by keeping them up-to-date.

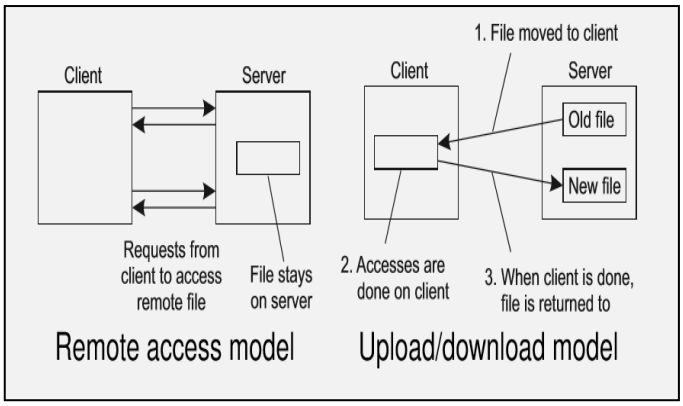


Fig. 1. Server-Clent Architecture Models

## II. Background

Network file systems [3][4] enable data to be accessed in a networking platform. In such a system, the data is stored on remote servers, and clients can access data by logging onto the server. All the changes made to a file by the user are propagated and saved on the server. A single server for multiple clients. distributed file systems have high reliability and capable of handling much more traffic.

### A. File Server Architecture

There are 3 basic components of the File Service Architecture namely, Flat File Services, Directory services and Client Module.

*1) Flat File service:* Uses unique file IDs for the requestsmade Selection

*2)* Directory Services: Converts human-readable names into unique file IDs. It is often used as the client in a Flat File Services.

*3)* Client Module: Interacts with other two. It also provides files to user programs.

Implementation of decentralized servers maintains multiple copies of the data. To make them fault tolerant, data replication is spread across different servers within the same data-centers and across data-centers at different geographical locations. Big companies such as Google and Facebook maintain a minimum of three copies of user data in their data-centers.

Modern distributed file systems also include simple RAID implementation of storing the files in a distributed fashion among the servers available. Corruption of data in any of the server or fault in any of the server acts as a single point failure for the entire distributed file system. To make the file system more reliable, fault tolerant techniques need to be applied on the file system to increase the robustness of the system.

In this paper, fault tolerance in Fuse File System is incorporated using data replication on multiple servers, where each data copy of the server is replicated in its next adjacent servers. A checksum comparison is performed between the data server copy and replica server copy to pick the correct data in case there is a corruption of one of the data.

## III. Implementataion

The system design is implemented in FUSE file system by making modifications to allocation of servers, dividing the file into blocks and storing the blocks onto two adjacent servers. When a file is corrupted or a server goes down, the file system on server returns the data from the replica copies and also tries to restore the correct value on the server containing the corrupted value.

### A. Server Allocation and Lookup

Whenever a server is to be allocated for writing a new file, a new server from within the server bank is allocated using round robin allocation method. The round robin allocation is dependent on the path of the file and not on the data present on the server. While reading a file, the client looks for the server the file is stored in.

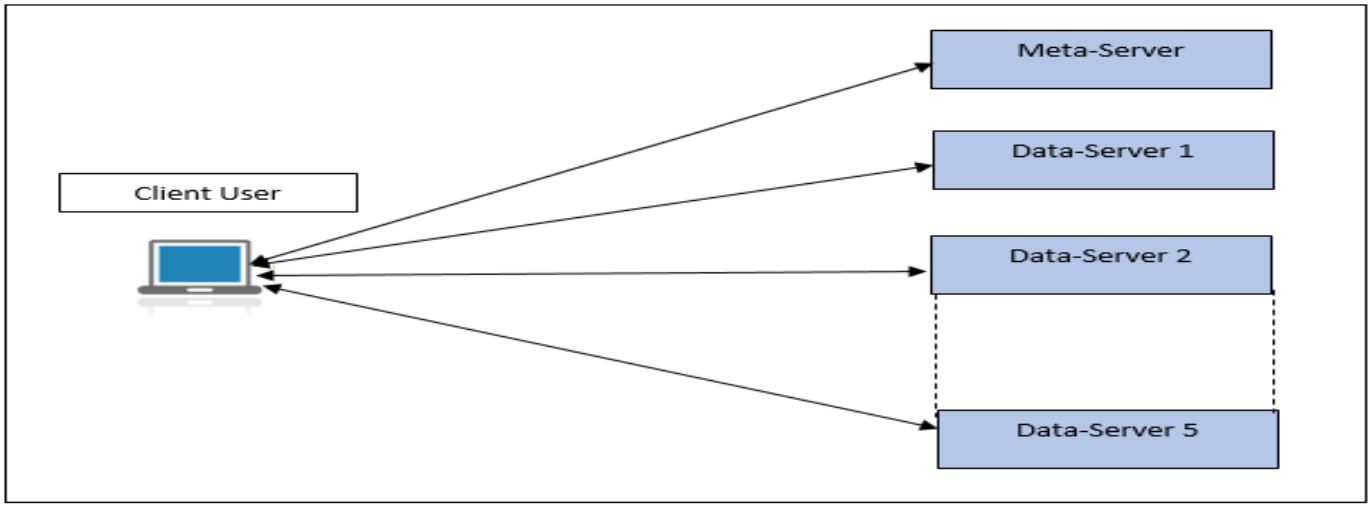


Fig. 2. Basic Design of the System

### B. *Reading data from server*

When a file is to be read, the meta-server which contains the list of those files which contains the data is pinged using the "get" function with appropriate attributes. If a ping "None" is obtained, then no file with the provided name exists on the server. There are two possibilities that can happen here on.

*1)* Data Server is Online: If the required servers are online, the data is requested using multiple "getdata" call to retieve all the assoicated blocks. On the server side, the data server initites an RPC call to the next adjacent server to perform check on the data.

*a)* We compare the precomputed checksum with the run-time checksum on data. On match, we do a sanitation check on the replica copy of the data on the next adjacent server.

*b)* If the previous condition doesn't match, the present server requests replica data from the next adjacent server and replace its corrupted data and returned.

*2)* Data Server is Offline/Crashed: Since, non-blocking read is allowed in our implemetation (under contraint;no adjacent server is offline) if the required servers are offline, the replica data is requested from the next adjacent data server.

### C. *Writing data to server*

When new file is written, the client will check whether that file exists in the file system by initiating a ping to the meta-server using "get" function, along with appropriate attributes. When it does not find an existing entry by that name, it will create a new entry for that filename after allocating the first write server. The first write server is determined by a hashing function which returns the sum of ASCII values of all the characters in path name of the file. Subsequent block allocation is governed by a round-robin fashion, across all the servers, using modulo arithmetic. Along with the data, replica copies of the blocks are also written into the next adjacent server.

To avoid inconsistency in the data, blocking write scheme is utilized. Blocking of write can be divided into two groups

*1)* Data server Online: If the required servers are online, all the writes are required to send an acknowlegment. Thereofore, every write to either data server and replica server does not return unless they are successful.

*2)* Data server Offline: If a server is offline, all the writes are blocked. The client will perform repeated attempts to write the data and will continue until the server comes online and sends an acknowlegment for the same.

### D. *Recovery and Restoration from Server Crash*

The system implementation provides support in form of two techniques namely; Recovery of data and Restoration of data.

*1)* Restoration: If the server crashes, the in-file system maintained by the server will be lost. In order to recover from this, a persistent file system is used in tandem with the in-file system. The

persistent file system follows a write-through policy. To recover from a crash, the in-memory system is build from the most rescent persistant file system availble for that server.

*2)* Recovery: If the server crashes and the persistant file system is lost, the system can still be bought to its previous state. The recovery process is initiated if the recover file is not found in the system. As the data in the previous server is the replica of the present server and the replica in the next server is the data in the present server, 2 RPC calls are made to the next and previous adjacent servers to restore the data-state of the server. While doing so, persistent file system is also build to prepare the server for the next server crash.

The name of the persistent file is given as "recover_data<server #>.txt"

### E. *De-clustering of files*

When a data is written, the data is first, divided into block of size given by "block_size" and then based on the hash function allocated to respective servers. The block division and use of hash function ensures that the data is balance and divided between the data-servers.

### F. *256-SHA Checksum*

256-bit Secure Hashing Algorithm is utilized to compute the checksum on the data. Although, md5 is much faster that the SHA 256 given it produces 128-bit output when compared to 256-bit from SHA, SHA256 provides benefits of less collision (theoretically) as the digest of SHA is larger than md5. Also, it provides a platform for security measures if we wish to extend the functionality to the system.

## IV. Testing Mechanism

The basic checks were performed to ensure the working of the network file system design. To test the functioning of the file system, six Server Proxy objects (5: data servers, 1: meta-servers) of each running server were created and accessed via a separate terminal window. These objects were used to exploit all the functions exposed by the server using the "put", "get", "getdata" and "putdata" interface by the client. The overview of testing is covered here

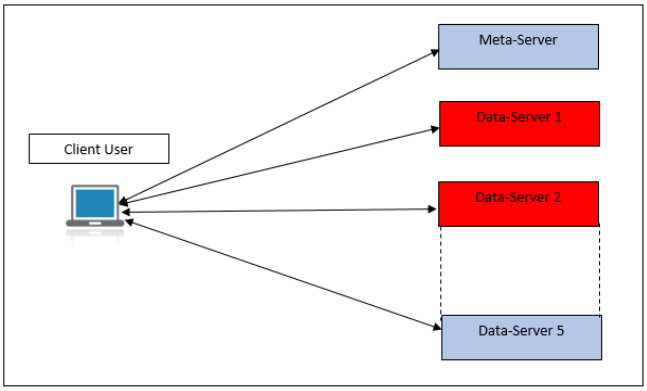


Fig. 3. Failure of System

### A. *Consistency of Data writing*

When data is written, every server must return an acknowledgement to return control to the client. This allows to keep the consistent data spread across the data servers. If the server does down between the process, the client retries to write the data till it gets an acknowledgement from the server.

### B. *Consistency of Data reading*

The lookup for a particular file is done using its filename as the key. Only the data blocks that are required for the read are requested by the client. If a sever is down on a read, data is read from the adjacent server.

### C. *Listing Metadata*

The attributes of all the files and directories in the file system is obtained from a single server that also saves the root ('/') directory. This server is called the metadata server and it contains the metadata for each of the listed files written on the data servers. The file system essentially fails completely when meta-server cannot be accessed by the client.

### D. *Server Crash*

Server crashes were tested for 2 conditions broadly.

*1)* Server crashes before/while read: The system is resilient if non adjacent servers crash. In order to read, the client issues a RPC call. If no data is returned in some time (Timeout), the client request the same data from the replica server. Testing for 3,4 and 5 data server was carried out, with multiple (2 servers) for 4 and 5 server cases.

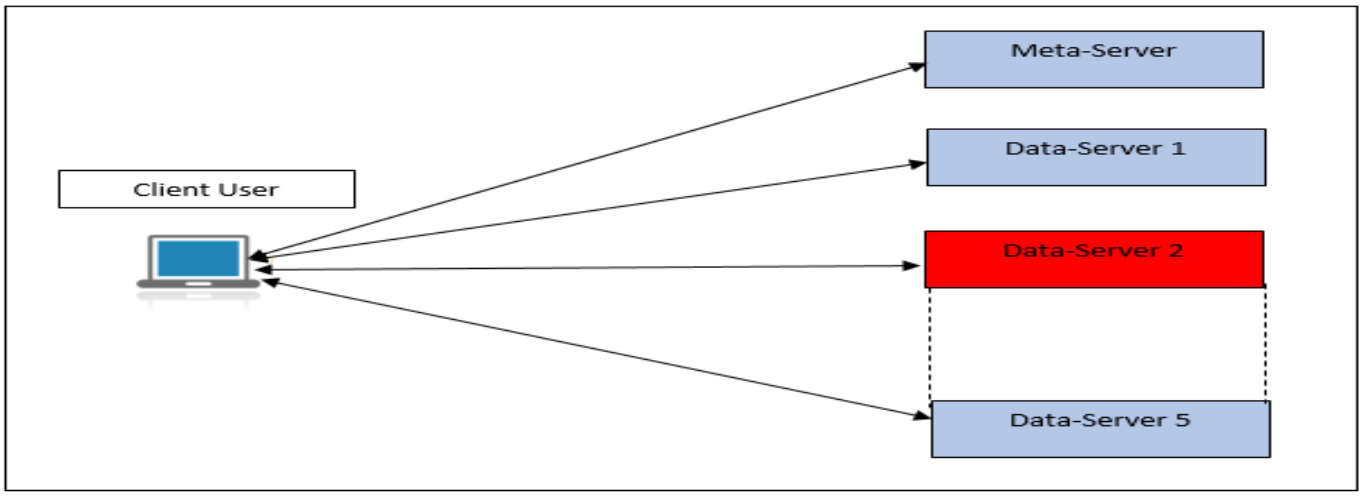


Fig. 4. Failure in one server

*2)* Server crashes before/while write: The write calls take the assumption that no non-idempodent values are written into the file-system. Keeping this assumption under check, the client tries to write in a server. Until an acknoledgement, before timeout, is received, the client retries to write the data for both data and replica server i.e. the write calls get blocked if the server to be written in down. For testing blocking write, servers proxy object were terminated and write operation were issued. As required, the system blocks the write call the client receives acknoledgement. Once the servers are resumed, the write operation completes. To verify the correctness of the data, standard UNIX commands like "cat" were used to see the data.

### *E. Data Corruption*

To test data robustness against corruption, separate API namely, "corrupt_data" was developed. This API's purpose is to initiate an RPC call to server and corrupt a data block. Using the file path as one of the attributes, the API picks up a block at random, initiates a RPC call to the associated server and corrupts the data.

Due to checks placed on the value, on the server side, the value returned by the data server is verified and compared with its replica copy. Since the data has been corrupted, as per our implementation scheme, the checksum fails to match leading to rewriting of the data fetched from the replica server into the present server.

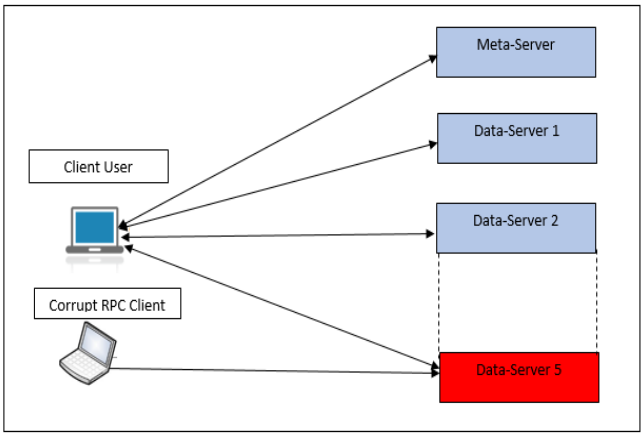


Fig. 5. Corruption of Data

Since, replica copies are considered golden standard under corruption, it becomes imperative that periodic sanitation check on the replica is performed. This feature has been incorporated into the read function of the server side. The function check the replica copy for corruption, and if required, send data to the replica server to overwrite corrupted data.

## V. CONCLUSION

The implementation of fault-tolerant remotely distributed file-system using upload/download model successfully meets our set goal. We successfully implemented fault tolerance in the data servers. The system design is capable to restore from corrupt data and server crashes during access,

successfully. It allows non-blocking reads even when non-adjacent servers are down along with blocking writes which allow for minimalistic implementation to achieve consistent memory data structure. The functionality is tested with results for each of the listed cases, which are reproducible.

## VI. SHORTFALLS

**Meta-Server Crashes**: All the attributes related to the file system are on a single meta-server. If the meta-server goes down, the entire distributed remote file system will be rendered inaccessible.

**Concurrent Modification**: Multiple users cannot modify the same file. To achieve it, we require concurrency and coherency model of cache blocks implementation.

## VII. FUTURE SCOPE

All the above listed limitation put focus on the need of a fault tolerant remote file system which not only provides a considerable robustness in read and write operation but, can withstand considerable number of failure of server nodes.

We can reduce network traffic by retaining recently accessed disk blocks in the cache so that repeated access to the same information can be handled locally. But, this will lead to cache consistency and coherence issues.

## VIII. WORK DIVISION

We divided the work equally and completed the project as a team. We planned the work division well ahead in time and helped each other throughout. We distributed the major part of coding as: Client side functionality and Server crashes by Suvrat and Server Side functionality and Data corruption by Siddharth. However, the integration, analysis, testing and bug fixing were done jointly.

## IX. ACKNOWLEDGMENT

We would like to thank Prof. Renato J. Figueiredo for giving us an opportunity to do this project work. We would also thank Mr. Jaikrishna Tanjore for his valuable input on the project.

## REFERENCES

[1] https://en.wikipedia.org/wiki/Fault_tolerance

[2] Peric, D.; Bocek, T.; Hecht, F.V.; Hausheer, D.; Stiller, B., "The Design and Evaluation of a Distributed Reliable File System," Parallel and Distributed Computing, Applications and Technologies, 2009 International Conference on, vol., no., pp.348,353, 8-11 Dec. 2009

[3] Joreme H. Saltzer, M. Frans Kaashoek “Principles of Computer System Design” ISBN: 978-0-12-374957-4

[4] Brian Pawlowski, Chet Juszczak, Peter Staubach, Carl Smith, Diane Lebel, and Dave Hitz. NFS version 3: Design and implementation. In Proceedings of the USENIX Summer 1994 Technical Conference, Boston, MA, USA, June 1994