Improving the reliability of a Network Filesystem by using Redundancy and Replication

Nikhil Pratap Ghanathe, Nachiket Gururaj Naik, Yathindra Kota

**Abstract:**

Designing a reliable system is a significant need in some areas rather than just optimizing it for peak performance. This requires considerable amount of resources as well as careful programming for synchronization. Fault tolerance is a technique of implementing a system where one or more errors or faults can be masked by the system and continue operation albeit with a reduced quality of service. Network filesystem is one such area where the data (which may be shared across many users) is mounted on the local host and accessed by one/multiple users. We aim to utilize a fault-tolerant approach for designing the network filesystem for maximizing data reliability while adding features that potentially boost the performance of the fault-tolerant filesystem.

**Problem Statement:**

The problem statement involves an implementation of a file system where the data server has to be replicated to improve performance and ensure fault tolerance. We have a single meta-data server and multiple data servers. Every file has multiple replicas with each data server having one copy of the file. The given file system needs to conform to the quorum approach, that is the files accept a read request only if the minimum number of servers are greater than or equal to the value of Qr and write request is accepted only if the value of Qw is equal to n replicas. The system also has to ensure fault tolerance with respect to various failure modes. It has to withstand issues like server crash, server restart or corruption of data in a server.

**Introduction:**

Ever since the dawn of the internet, high bandwidth broadband, smart phones and various other technology and devices, we are online most of the time with a constant requirement for the data to be delivered directly to our personal computers, smartphones and other similar devices. Applications services like web search, emails, instant messaging, gaming all of them need to have data transfer at high speeds. Exchange of data electronically is regularly required for almost all types of business interactions, government and banking applications.  With this huge demand for almost instantaneous delivery of digital information, there arises the need to have concentrations of computer and connecting network equipment that is capable of handling these requests and provide service. Thus, lead to the birth of modern data centers.

A data center is a centralized facility in which the organization’s computing and networking equipment is placed together with the intention of gathering, managing, storing and disseminating large amounts of data. These house the most important systems of an organization and are critical for the continuity of the everyday operation of the organization. Therefore a data center must maintain very high standards of assurance with regards to its integrity and functionality in the system it has been hosted. Efforts must be made to keep the datacenter working at all times as any failure with the data center will lead to impairment of majority of the organization’s operations.

Fault Tolerance of a system is that ability which allows it to function properly even in the event of several of its components failing. It is critical property in system which are of high-availability or which are life-critical. The system, if, designed to be fault-tolerant is able to sustain its operations, possibly at a lower level than completely failing when a part of the system fails.

**Motivation:**

The system failure can happen due to two main reasons, one being the failure of data center itself, the other being the failure of the network which transmits the data from data-center to the clients.

Data center failures can occur due to many reasons, few of them are:

1. **Data center internal failures**: The main reason for the data center down time is related to the power supply to the system. The uninterrupted power supply (UPS) system battery failure, UPS capacity exceeding the operating voltages, UPS equipment failures amount to major share of the unplanned data center down times. The study by Ponemon institute [2] infers that over one failure occur every year, resulting in average down time of 91 minutes.
2. **Natural Disasters**: Inter data centers are vulnerable to natural disasters like hurricanes, earth quake, tsunami etc. 60 enterprise data centers were affected when the Sichuan earthquake occurred in 2008. 10+ data centers were brought down when Tohoku earthquake and tsunami occurred. This can even lead for the companies to go bankrupt. To avoid this and for having sufficient data redundancy, distributed redundant servers are needed. [1]
3. **Cyber attacks** also result in significant number of server failures. This reason for the failure is increasing more than any other data-center failure causes. This has increased from 15% in 2010 to 35% in 2013. **[2]**
4. **Human errors**, although a small percentage also is one of the reasons for failure.

The failure of networks is also a main reason of the system failure. The network failures occur due to many reasons, some of them being,

1. Faults, errors in the network devices
2. Power outages
3. Unprecedented natural disasters on network such as minor accidents, extreme weather conditions etc.
4. Link failure due to fiber cable cuts or network congestion
5. Like in Data center failure, simple human errors.

Cloud services delivered by a data center provides immense opportunity for online gaming, social networking, video on demand etc. Customer experience is vital for these applications. Recent studies point out that, in these services, if the system takes more than 2s to load, the users will start to abandon the system. This increases by 5.8% with every second increase. 60% of the total abandoners never will come back. [**3]**

If the server-network system is in series, the entire system fails if either of the two fail. To avoid this, system redundancy is needed.

Above mentioned reasons are good enough for the need of redundancy in the systems. In our report, we have implemented a system using multiple servers which achieves the goal of providing un-interrupted service to the clients.

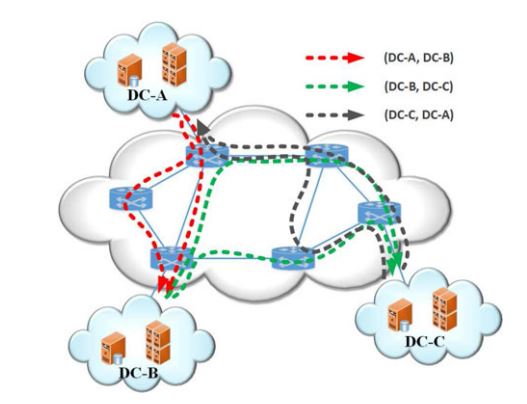


Image Source: [1]

**Methodology:**

Our implementation methodology involves, running multiple data servers by spawning multiple processes. The calls to these servers are sent via XML-RPC protocol after sending it through a hash function generator. The data is written onto all the servers, and it is read from at-least three servers as per the Quorum approach. Multiple cases like server restart, server crash and other such scenarios were taken into account while designing the system.

**Benefits:**

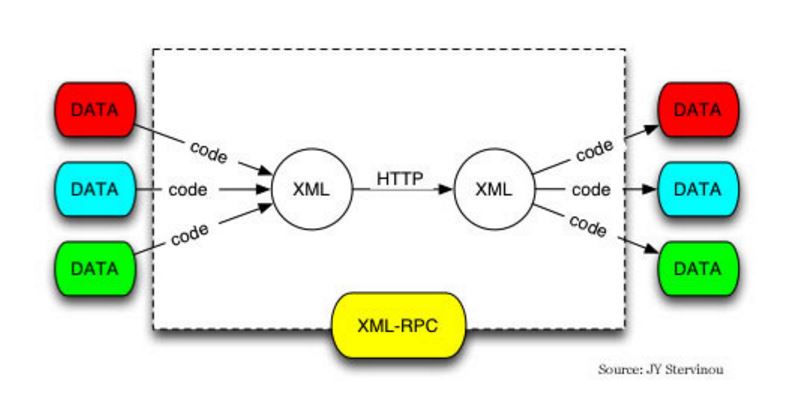
As we saw above, the loss of data and server downtime is a major concern in data-storage domain. A simple system is designed and is implemented which can tolerate data errors, server crash and other such scenarios. The system is tested for multiple test scenarios which are of big concern in the field of data storage. A list of possible improvements are analyzed and added in this report.

**Background:**

A few python libraries and concepts were used in the implementation. Some of which are explained below.

1. **XML-RPC:**

XML-Remote Procedure Call (RPC) works on the data received in the transport layer of the computer network. As the name suggests, this protocol is used to call a procedure in remote system. It works by sending data through HTTP request to server implementing the protocol. XML-RPC uses a separate set of XML vocabulary to describe the protocol of requests and responses. XML-RPC has 6 basic data types which are used in the communication. They are int, double, bool, string, datetime, base64.



**Image Source**: http://webkul.com/blog/openerp-7-0-connection-with-php-using-xml-rpc/

1. **MD5 protocol:**

Message digest algorithm is a 128-bit has function generator. This is very commonly in verifying the data integrity in the data-transfer applications. This is widely used to provide assurance of the integrity of the files. For example, file transfer applications usually use md5 checksum for files so that the integrity of files can be verified after download.

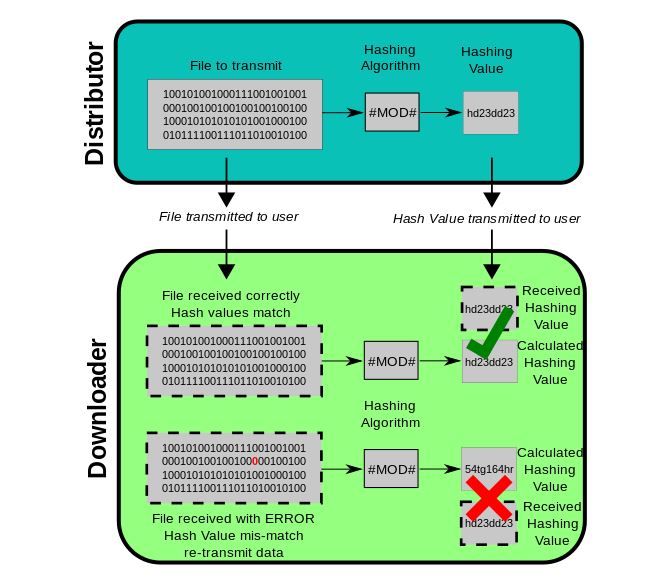


Image source: <https://en.wikipedia.org/wiki/MD5>

In python, the library md5 can be used to implement the md5 functions. It creates a 128 bit hash value which is different for any given file. Two files with different data can be said to have different hash value with a high probability.

1. **Quorum approach:**

In any distributed system containing replicated servers or databases, a quorum is the minimum number of votes that any particular operation has to receive in order to complete the operation. For any operations which consist of data read or data write, we have a read quorum (Qr) and a write quorum (Qw) which needs to be satisfied to ensure read or write of data. The read and write quorums should follow the rule Qr + Qw > Nreplicas. Having the sum of read and write quorums greater than the number of servers, ensures that read quorum consists of at the worst case, one server having the latest version of the data. By managing the values of Qr and Qw, we can manage availability in favor of read or write as required. In this particular implementation, we have been given the values of write quorum Qw= Nreplicas and Qr >=3

**Implementation:**

Our implementation consists of a Meta data server, multiple data servers the value of which is defined by the number of ports entered through the command line. The connection between the servers and the client is established by the XML-RPC protocol.

The data storage is done in the following way. On creation of a file for instance, the file contains three major components-the meta data, the list-nodes, and the actual data. The first two components, the metadata and the listnodes are stored in the meta server while the data is stored in multiple data servers along with an error check sum for error detection. They are stored as a key-value combination with the key containing the path of the file while the value is the actual data. While the data is requested, the data is received from one server and checked for the check sum. If it matches with the original check sum, data is delivered else a voting mechanism is involved to determine the right data which is then sent to the client.

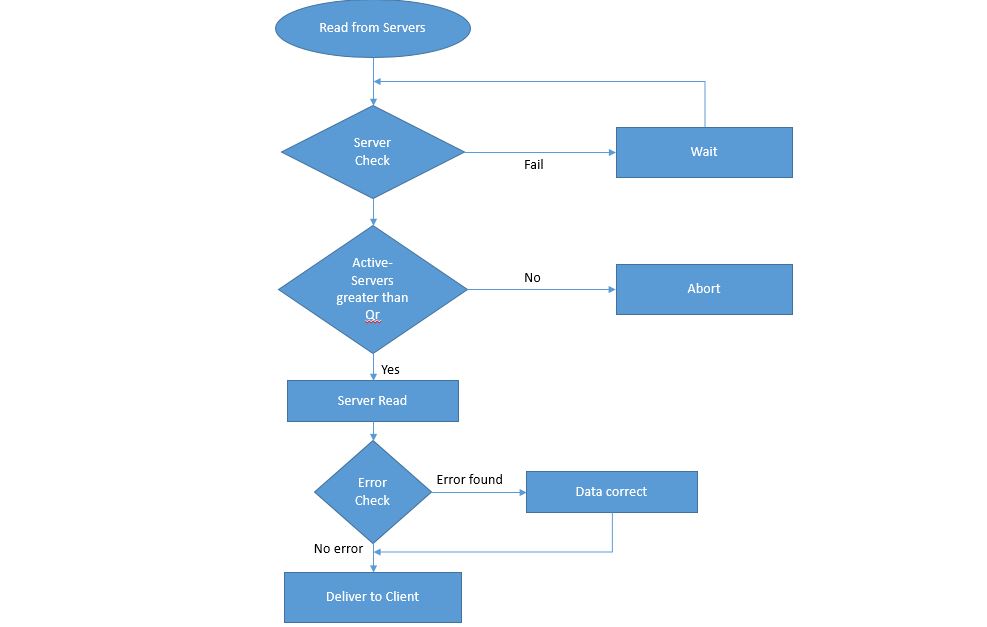
1. **Get method:**

This method is used to perform the read operation. The client requests for the already existing data in the data server. The data is returned to the user if the data is received from at least Qr number of servers. If the data received by this method is different from a server, the data is modified in the server to a correct value and finally returned to the user. Detailed explanation is given below.

1. **MD5 functions:**

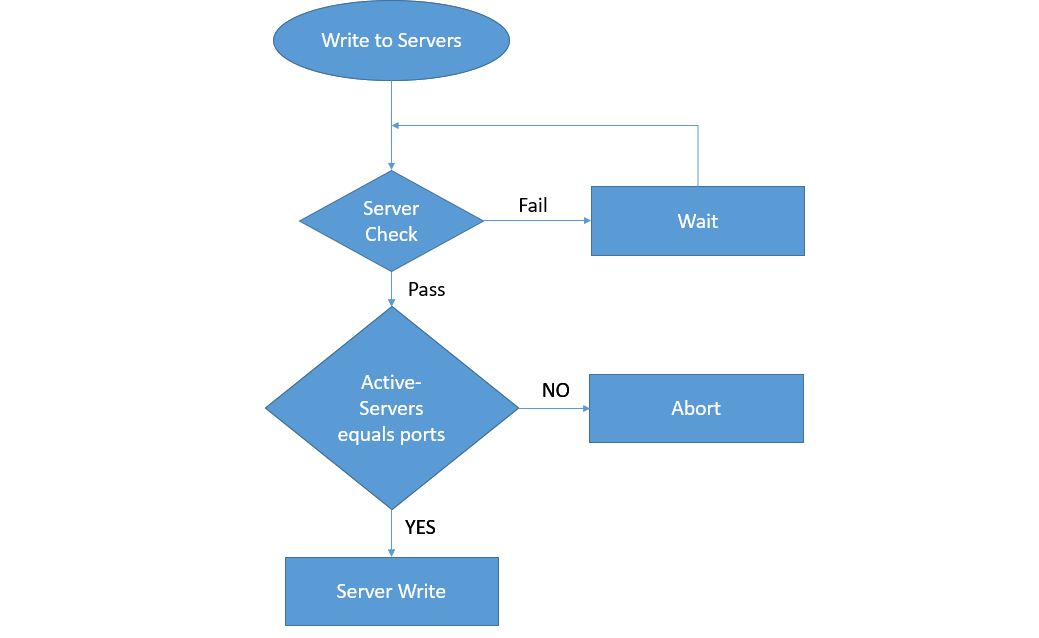
Using the MD5 python library, a hash is generated for all the files which need to be written on the data servers. When a file is retrieved from the data servers, the hash data is extracted from the files and the correctness of data is checked.

In the get method, we initially check the status of the servers (server\_check) for their availability. If the list of active servers (servers not powered off) is greater than the number specified by the value of Qr (Read Quorum) from the command line, the data from all server is obtained. If the number of active servers are less than the value of Qr, we do not complete the get operation as the quorum approach forbids the read operation if the number of active servers is less than Qr. Now we examine data from each server and store the particular data of file into a list. While performing the get function, if we are not able to connect to a server we wait for a period of 10 seconds and try again. If we don’t get a response again, we stop trying and proceed to the next server. The list obtained is then sent to the voting mechanism (error\_check) from which we obtain the correct data which is then transferred to the client.



1. **Put method:**

This method is used by the client to write data to a particular file in the server. In the put method, we first check the active status of each of the server (server\_check), during the check if we find out about an inactive or switched-off server, we abort the write operation as the quorum approach does not allow writing of data into a file if there is even one inactive server. If all the servers are active we go ahead and try putting the data in all the servers with the path name as the key. If we are unable to put data into a particular server the first time, we wait for a period of 10 seconds and then try again. If we are still unable to write into that particular data server we abandon the write of that data according to the quorum.

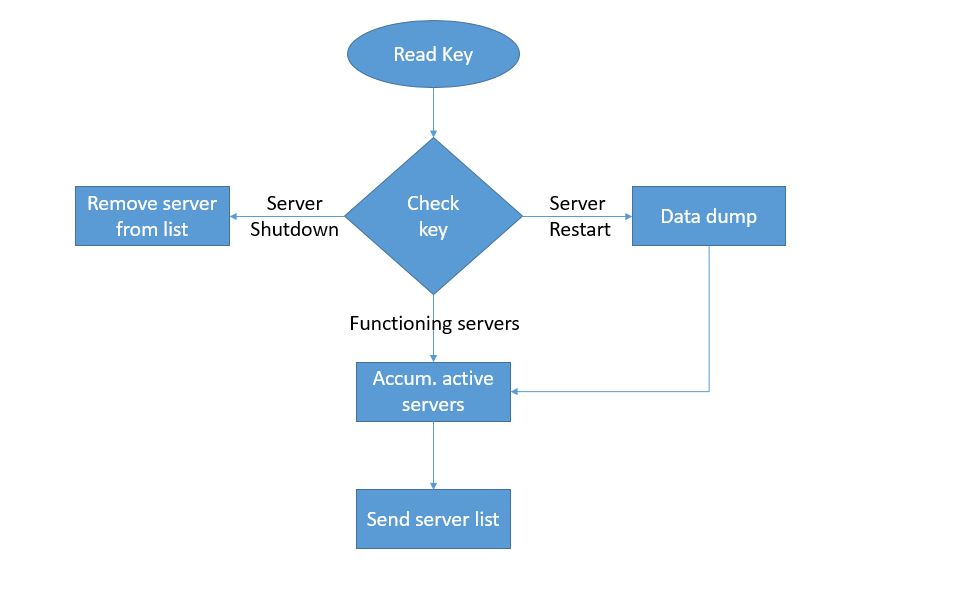


1. **Server check method:**

This method is used to check the status of each server. Here we determine the server state which determine whether a particular read or write operation should be performed based on the quorum approach. This method is performed before every get or put operation. In the put operation, we only check the number of servers active. If a particular server does not respond for the first time, we wait for a period of 10 seconds and then retry. If we still end up not getting the response we conclude the particular server is inactive. This method returns the list of all the active servers to the put method which then decides if it wants to continue its write operation based on the quorum.

The server check method for the get method is a little more complicated. It first involves the check of the existing servers. The key used for the get operation which is basically the path of the file is also used in this method. We check the existence of the key in the server. If a key is not present it basically means two things, either the server is shut down permanently or it had shut down and restarted with all its memory lost. The check\_key() method returns ‘true’ if the key is present a value ‘false’ if the server does not have the key and throws an exception ‘Connection Refused’ if the server is shut down. If the server is shutdown, we wait for 10 seconds and retry. If we do not get any response, we assume the server is shutdown permanently.

For the case in which the return value is false, we understand that the server is working but has lost all its memory after restart. So we handle this situation by calling the server\_dump() method. In the server\_dump() method, we check for the first active server and then transfer completely the server data contents into the server which had lost its memory. Thus we now have all active servers with complete data. This list of active servers are sent back as a list to the get method which decides it’s next course of action based on the quorum.



1. **Error Check method:**

This method is used to determine the correct value that has to be passed to the client. On request of a particular data from the client, all the data servers provide their corresponding data. This method is used as a check to ensure that no data corrupted is passed to the client. Here we have implemented a voting mechanism where in the data from all the servers are compared and the data present in majority of the servers are sent to the client. We have an assumption that data from only one server can be corrupted, hence we check for the unique values. There can be only two possibilities, the first case when all values are identical. Here we do nothing and take the value from any of the servers and pass it to the client. The second case is when all but one servers provide identical data with one corrupt data. Here we single out the corrupt data, send it to the error\_correct() method. The error\_correct() method corrects the corrupt data and we now have a set of identical data in all servers and one of which is sent to the client.

**Potential Improvements:**

* By using the data obtained through the MD5 has functions from one data server and in tandem checking the data from the other servers (to satisfy Quorum requirement), the performance can be improved considerably.
* In our implementation, multiple processes are spawned to open multiple data servers. The same approach can be used to perform multiple operations on the data servers at the same time. This helps in improving the read performance of the system.
* Currently the server data-dump to a restarted server is performed only when a user performs an operation. If the same is continuously checked through a middle layer, the servers can be kept active and up-to-date with all the operations performed. A subsequent read/write operation does not have to wait for the server-dump to happen.

**Test Cases:**

For the purpose of evaluation, a python test script is included. This section will describe the functions in the test case that evaluate the performance and primarily the functionality of the filesystem.

1. **test\_01\_NestedDirectories**: This function tests the basic functionalities of the filesystem like creating directories in various levels of hierarchy in the filesystem including nested directories.
2. **test\_02\_NestedFileCreation**: This function tests the nested creation of files and tests whether they are present as created.
3. **test\_03\_ReadFiles**: Reads various files which were previously created.
4. **test\_04\_removeFile**: Tests the ‘rm’ functionality i.e. removal/deletion of files
5. **test\_05\_removeDir**: Tests the ‘rmdir’ functionality i.e. removal/deletion of directories
6. **test\_06\_renameFile**: Tests ‘mv’ functionality i.e. renaming of files and directories
7. **test\_07\_corruptTest**: This function tests the error correction capability of the filesystem. It first writes some data into a file. Then the data copy of a particular server is corrupted. Now again the file is read. During the read, the system detects the error and corrects it. Finally it returns the correct value.
8. **test\_08\_terminateTestRead**: This function terminates a server and then proceeds to create and write a file. This does not happen as the Qw Quorum is not satisfied. Hence no file is created. When we try to read this file, we get a **I/O error : No Such file or directory** signifying that the files was never created. This is the expected result.
9. **test\_09\_terminateTestWriteAfterRestart**: This function restarts the terminated in the previously terminated server. After restart a file is created and written into. This is subsequently read and it returns the correct value. This is because the number of servers are now equal to Qw. Hence the write should complete, and it does.
10. **test\_10\_terminateTestRead**: This function tests the fault tolerance on read operation. Assuming we have 5 servers running, and Qr>=3, the read function returns correct value till we terminate 2 servers. When we terminate the 3rd server, Qr<3, hence the read fails and an assertion error is seen on the screen.

Hence after we run the test script, all the test should pass except **test\_08\_terminateTestRead** which will throw an I/O error: No such file or directory and **test\_10\_terminateTestRead** which will fail when number of servers is not equal to Qr. This is the expected output. The time to run 10 tests is **12.163s.**

**Results:**

The filesystem was implemented and is designed to be fault-tolerant till the Quorum specification is met. The system also fails gracefully when the Quorum specification is not met. The system was subjected to rigorous testing simulating all possibilities of faults and failures. It is observed that the devised system is fault-tolerant and masks errors to an extent specified by the **Qr** and **Qw** value specified on the client side.

Following are some of the performance results of read operation performed on the servers. The execution times indicate the masking and error detection of fault tolerant filesystem.

|  |  |
| --- | --- |
| **Operations** | **Execution time (s)** |
| Read when all servers are up and running with no data corruption | 0.0511028766632 |
| Read when one of the servers is corrupted | 0.0548038482666 |
| Read when a server is restarted and data has to filled in this restarted server | 0.0548329353333 |

**Bibliography:**

[1] Jingjing Yao; Ping Lu; Long Gong; Zuqing Zhu “On Fast and Coordinated Data Backup in Geo-Distributed Optical Inter-Datacenter Networks” Journal of Lightwave Technology

[2]http://www.emersonnetworkpower.com/documentation/enus/brands/liebert/documents/white%20papers/2013\_emerson\_data\_center\_outages\_sl-24679.pdf

[3] A. Krishnan and R. Sitaraman, “Video stream quality impacts viewer behavior: Inferring causality using quasi-experimental designs,” in Proc. Internet Meas. Conf., Nov. 2012, pp. 1–14

[4] Chuanxiong Guo, Haitao Wu, Kun Tan, Lei Shi, Yongguang Zhang, Songwu Lu “DCell: A Scalable and Fault-Tolerant Network Structure for Data Centers”

[5] Yang Liu, Dong Lin, Jogesh Muppala, Mounir Hamdi, “A Study of Fault-tolerance Characteristics of Data Center Networks”