Int. J. Communications, Network and System Sciences, 2019, 12, 49-58
http://www.scirp.org/journal/ijcns
ISSN Online: 1913-3723
ISSN Print: 1913-3715

Scalable Distributed File Sharing System: A Robust Strategy for a Reliable Networked Environment in Tertiary Institutions

Emmanuel N. Ekwonwune[1], Bright U. Ezeoha[2]

[1]Department of Computer Science, Imo State University, Owerri, Nigeria
[2]Department of Computer Science, Abia State Polytechnic, Aba, Nigeria
Email: ndukwe@abiastatepolytechnic.edu.ng, bright.ezeoha@abiastatepolytechnic.edu.ng

How to cite this paper: Ekwonwune, E.N. and Ezeoha, B.U. (2019) Scalable Distributed File Sharing System: A Robust Strategy for a Reliable Networked Environment in Tertiary Institutions. *Int. J. Communications, Network and System Sciences*, **12**, 49-58.
https://doi.org/10.4236/ijcns.2019.124005

Received: October 15, 2018
Accepted: April 27, 2019
Published: April 30, 2019

Abstract

The bane of achieving a scalable distributed file sharing system is the centralized data system and single server oriented file [sharing] system. In this paper, the solution to the problems in a distributed environment is presented. Thus, inability to upload sizeable files, slow transportation of files, weak security and lack of fault tolerance are the major problems in the existing system. Hence, the utmost need is to build a client-server network that runs on two or more server systems in order to implement scalability, such that when one server is down, the other(s) would still hold up the activities within the network. And to achieve this reliable network environment, LINUX network operating system is recommended among others as a preferred platform for the synchronization of the server systems, such that every user activity like sending of internal memos/mails, publication of academic articles, is replicated; thereby, achieving the proposed result. Hence, Scalable Distributed File Sharing System provides the robustness required to having a reliable intranet.

Keywords

Distributed System, Architecture, Files Sharing, Distributed File System, Replication, Reliability, Transparency, Data Access Interfaces, Fault Detection, Fault Tolerance, Cache Consistency, Scalability

1. Introduction

Information and its security are what drive the speed of technological inventions in this contemporary age—Information Technology Age. Hence, technology has penetrated into every facet of living which includes education, commerce, mili-

tary, health, law, etc. The internet as a tool offers the possibility of global information sharing and collaboration [1]. Computer storage is an increasingly important part of the internet; hence, ensuring the security and integrity of stored data is a crucial need. Therefore, it has been observed that distributed file sharing system is poorly embraced and or applied in developing countries such as Nigeria, unlike in global and multi-national companies like Google that have predominantly developed its full concept. Nigerian academic institutions are not left out in this seeming prevailing challenge, considering the regular growth and movement of data there: the full automation of academic activities such as the dissemination of internal memos/mails and research papers/articles within the school-owned distributed environment (local area network) has become a nearly impossible task because of the challenges distributed file system presents.

However, a closer study of this "all important" technological invention, Distributed File System (DFS) shows that it presents a variety of challenges that cannot be ignored. Though, it is seen as and has become a normal part of our daily life, it presents scalability problem (especially when a server is to be mounted or dismounted) and slow transportation of files because of traffic and security. It is obvious that attacks by intruders and insiders have led to billions of naira in lost revenue and expended effort to fix the problems. Most organizations, some academic institutions inclusive, rely heavily on their distributed computing environment, which usually consists of workstations and a shared file system. This file system is often stored on a "centralized file server" that is managed by a system administrator who has access to the whole file system, leaving the data vulnerable to anyone who can prove (legitimacy or otherwise) that he is the administrator. Recently also, network-attached disks have begun to replace traditional centralized storage systems. In such systems, disks are attached directly to a network and rely upon their own security rather than the server's protection. Yet, this arrangement makes security more difficult because the disk is directly exposed to potential attacks instead of being hidden behind a single server.

Furthermore, existing file sharing system and their access control models have been challenged by the scale and administrative complexity of the internet. Example of such a system is the network file sharing (NFS). And such systems share the property that a relatively small set of users have read/write access to files as current access control systems rely on authentication requiring that a user is known to the system [1].

More so, it is obvious that no client should be concerned or affected by back-end issues (*i.e.* server faults and activities such as mounting new, and dismounting of existing server). Unfortunately, this problem is the order of the day. Once a server fails within an intranet, every activity at all clients ends halts; hence, a better and lasting solution to these resultant challenges is in urgent need. And Scalable Distributed File Sharing System is the answer.

This paper, therefore, recommends the use of two or more servers in order to

achieve scalability of the Distributed File Sharing System with LINUX network operating system. Thus, the institutional workflow will not be halted when a server goes down because there is a fall back server which keeps the network system/environment running.

This paper has been organized as follows: Section 1 contains the Introduction for the needed background knowledge, Section 2 talks about the Theoretical Framework based on the views of other authors about this research area, Section 3 contains the Summary of this paper, Conclusion is contained in Section 4, and Section 5 is the Recommendation.

2. Theoretical Framework

File sharing can mean distribution of access to physical or electronic files. A good example is the transferring of documents or files to certain staff of an organization for a common purpose either by hand or storage devices such as flash disk, CD-ROM, or hard disk.

However, file sharing almost always means sharing files in a network, even if it is in a small local area network (LAN) [2]. It is the practice of distributing or providing access to digital media, such as computer programs, multimedia (audio, images and video), documents, or electronic books [3]. It is the public or private sharing of computer data or space in a network with various levels of access privilege [2]. Hence, file sharing involves two or more persons (computers, better put) to access or use a file either to read or write, or both. Therefore, it highlights the practice of sharing or offering access to digital information or resources, including documents, multimedia (audio/video), graphics, computer programs, images and e-books. It is the private or public distribution of data or resources in a network with different levels of sharing privileges.

Distributed File System (DFS) supports the sharing of information in the form of files throughout the intranet. And it allows users to store and access remote files like in a local way, but from any computer within the intranet. Thus, it is a client/server-based application that allows clients to access and process data stored on the server as if it were on their own computer [4] [5]. When a user accesses a file on the server, the server sends the user a copy of the file, which is cached on the user's computer while the data is being processed and is then returned to the server. Ideally, a distributed file system organizes file and directory services of individual servers into a global directory in such a way that remote data access is not location specific but is identical from any client. And all files are made accessible to all users of the global file system, and the organization is hierarchical and directory based. It offers us a file system that stores its data on a server, and its data can be accessed and used as though it were on a local node. The Distributed File System makes it convenient to share information and files among users on a network in a controlled and authorized way. And the server allows the client users to share files and store data just like they are storing the information locally. However, the server has full control over the data and gives

access control to the clients.

2.1. Factors that Influence Performance of Distributed Files System over Traditional Client/Server System

The purpose of a distributed file system (DFS) is to allow users of physically distributed computers to share data and storage resources by using a common file system [5]. And, unlike the traditional client/server solutions, better performance can be achieved in a distributed file system. And factors that influence the better performance of distributed file system include:

1) Instead of storing data on a single server, data can be stored on several nodes. This is known as replication.

2) The system is always available each moment a client connects to it. This is reliability.

3) The system has the capability to serve the clients' request at every log in an instance. This is referred to as availability.

4) Data is not lost but secured given that it is not stored on a single server.

Compared to a traditional client/server system where the data are stored on one server, the unique security feature of a distributed file system is that important or frequently required data in distributed file system can be stored on several nodes (nodes means a computer operating in a Distributed File System) [6]. This is called "replication". Replication can be used for achieving better system performance and, or "reliability" of the system. The data in a distributed file system are more protected from a node failure. If one or more nodes fail, other nodes are able to provide all functionality. This property is also known as "availability" or "reliability". The difference between availability and reliability is simple: availability means that the system can serve clients request at a moment when the client connects to the system. Reliability means that the system is available all the time when the clients connect to it. Files can also be moved among nodes. This is typically invoked by an administrator, and it is done for improving a load balancing among nodes. The users should be unaware of where the services are located and also the transferring from a local machine to a remote one should also be transparent. In a distributed file system, this property is known as transparency. If the capacity of the nodes is not enough for storing files, new nodes can be added to the existing distributed file system to increase its capacity. And this is known as "scalability". A client usually communicates with the distributed file system using the local area network (LAN).

2.2. Key Features of a Distributed File System

A distributed file system is expected to exhibit three basic features that will ensure reliable and secured file sharing environment among many others; namely transparency, fault-tolerance, and scalability.

1) Transparency: Users should access the system regardless of where they log in, be able to perform the same operations on the distributed file system and lo-

cal file system, and should not care about faults because of the distributed nature of the file system; thanks to fault tolerance mechanisms. Also, Transparency can be viewed in terms of performance. In this way, data manipulations should be at least as efficient as on conventional file systems. In other words, the complexity of the underlying system must be hidden to users: The end-user does not need to know how the system is designed, how data is to be located and accessed, and how faults are detected [7]. And the features that ensure transparency are:

a) Naming: this is a mapping between a local name and a physical location of a data. For example, in a classic file system, clients use logical name (textual name) to access a file which is mapped to physical disk blocks. And in a distributed file system, server's names holding the disk on which data is stored must be added. The distributed file system must respect location transparency: details of how and where files are stored are hidden to clients. Furthermore, multiple copies of files may exist, so mapping must return a set of locations of all of the available copies. The distributed file system should be location independent: the logical name should not change even if the file is moved to another physical location. To do so, allocation tables or sophisticated algorithms are used to provide a global namespace structure that is the same namespace for all clients.

b) Data Access Interfaces: Clients can create, read, write, delete files without thinking about the complex mechanisms of the underlying system which performs operation and must be provided with an access to the system with the help of simple tools. Here are some examples of such tools:

i) Command Line Interface (CLI) which is used to access files with traditional UNIX command.

ii) Java, CC^{++}, other programing languages and REST (Web-based) API can be used for graphic interface like the window explorer.

iii) Users can be allowed to mount (attach) remote directories to their local file system: thus, accessing remote files as if they were stored in a local device. The FUSE mechanism or the UNIX mount command are some examples.

c) Caching: This is a technique which consists in temporally storing requested data into the client's memory. Distributed file systems use caching to avoid additional network traffic and CPU consumption caused by repeated queries on the same file and thus, increase performance. When a data is required for the first time, a copy is made from the server that holds this to the client's main memory. Thus, for every future request of this data, the client will use the local copy, avoiding communication with the server and disk access. This feature is related to performance transparency since requests can be quickly performed, hiding data distribution to users with this technique. However, when a data is changed the modification must be propagated to the server and to any other client that has cached the data [7].

Thus, a cache in the computer system is seen as a component which stores data that were requested, hence can be potentially used in the future [6]. When the cached data are requested, the response time is shorter than when the data

are not in a cache and must be downloaded. The cached data can be stored in RAM for fast access and/or on Hard disk. A cache can be on both sides of communication. On the server side, the cache is usually located in RAM. On the client side, the cache can be located in RAM or on hard disk. On the server side, if file content is cached, cache mechanism spares time because there is no need to access file content from hard disk. On the client side, if the client requests file, cache mechanisms spare time because there is no need to communicate with a server. Client-side caching is also sometimes called client initiated replication. Client cache can also provide so-called offline cache. This means that the client can access the file from cache after disconnection from the server. Offline cache is often stored on hard disk; offline caching mechanism is used in CODA.

d) Fault Detection: A fault-tolerant system should not be stopped in case of transient or partial failures. Faults considered are network and server failures that make data services unavailable, data integrity and consistency when several users concurrently access data. In other words, this is the ability to detect overloaded servers, correct behavior of a server or corrupted data, and make decision to correct these faults. In a distributed file system, faults must be detected by the system, using minimum resources before being corrected, so that users would not be aware that such faults occur. All machines communicate together in a transparent manner by exchanging small messages. For example, the report allows servers managing the namespace to know what data are held by which sever. Since data are always in movement, this allows the system to identify which data is lost and needs to be moved or recopied when a server becomes unavailable or overloaded. Another message is heartbeats which are used to confirm the server's availability. If one does not send heartbeats for a time, it is moved to quarantine, and report messages are used to apply the correct decision [7].

2) Fault Tolerance: Faults considered are network and server failures that make data and services unavailable, data integrity and consistency when several users concurrently access data. Therefore, a fault-tolerant system should not be stopped in case of these transient or partial failures: In distributed file systems, network and server failure are the norms rather than the exception. Tools must be deployed to maintain and ensure that data are always available, to guarantee query processing in case of faults. Integrity and consistency of data must also be taken into account since mechanisms like caching or replication are provided.

Moreover, some features to ensure fault tolerance are:

a) Replication and Placement Policy: in order to make data always available, even if a server crashes, distributed file systems use replication of files by making several copies of data on different servers. When a client requests a data, he transparently accesses one of the copies. To improve fault tolerance, replicas are stored on different servers according to a placement policy. For example, replicas can be stored on different nodes, on different tracks, at different geographical locations, so that if a fault occurs anywhere in the system, data is still available.

b) Synchronization: in distributed file systems, synchronization between copies of data must be taken into account. When a data is rewritten, all of its copies must be updated to provide users with the latest version of the data. Three main approaches exist:

i) In the synchronous method, any request on modified data is blocked until all the copies are updated. This ensures the users access the latest version of the data, but delays query executions.

ii) In the second method called asynchronous, requests on modified data are allowed, even if copies are not updated. This way, requests could be performed in a reasonable time, but users can access an out-of-date copy.

iii) The last approach is a trade-off between the first two in the semi-asynchronous method, requests are blocked until some copies, but not all, are updated. For example, let's assume there are file copies of a data, a request on this data will be allowed once three copies will be updated. This limits the possibility to access out-of-date data while reducing delay for query executions.

c) Cache Consistency: this is the same problem as synchronization—how to update all copies of a data in cache when one of them is modified. Data can be cached to improve the performance of the system which can lead to inconsistencies between copies when one of them is changed by a user. These modification needs to be propagated to all copies and data in cache to provide users an up-to-date version of them. To avoid this problem, different approaches are used:

i) Write Only/Read Many (WORM): this is the first approach to ensure consistency only. Once a file is created, it cannot be modified. Cached files are in read-only mode. Therefore, each read reflects the latest version of the data.

ii) A second method is transactional locking which consists in obtaining a read lock on the request data, so that any other user cannot perform or write on data, or updating a write lock in order to prevent any reads or writes on the disk. Therefore, each read reflects the latest write, and each write is done in order.

iii) Another Approach Is Leasing: It is a contract for a limited period between the server holding the data and the client requesting this data for writing. The lease is provided when the data is requested, and during the lease the client is guaranteed that no other user can modify the data. The data is available again if the lease expires or if the client releases its right. For future read requests, the cache is updated if the data has been modified. For future write requests, a lease is provided to the client if allowed (that is, no lease exists for this data or the rights are released).

d) Load Balancing: this is the ability to auto-balance the system after adding or removing servers. Tools must be provided to recover lost data, to store them on other servers or to move them from a host device to a newly added one. Communication between machines allows the system to detect server failures and server overload. And to correct these faults, servers can be added or removed. When a server is removed from the system, the later must be able to recover the lost data and to store them on other servers. When a server is added to

the system, tools for moving data from a host server to the newly added server must be provided. Users do not have to be aware of this mechanism. Usually, distributed file systems use a scheduled list in which they put data to be moved or recopied. Periodically, an algorithm iterates over this list and performs the deserved action. For example, CEPH uses a function called "Controlled Replication under Scalable Hashing" (CRUSH) to randomly store new data, move a subject of existing data to new storage resources and uniformly restore data from removed storage resources.

3) Scalability: This is the ability to efficiently leverage large amounts of servers which are dynamically and continuously added in the system [7]. Contrary to common knowledge of a distributed file system, this implies that this system can involve more than one server within a Local Area Network for a better performance in file sharing. A practical instance is a situation where two or more servers are used within a networked environment: when one server in the intranet is down, operations and services of the distributed file system are expected not to stop but be synchronized and virtually transferred to the other server(s); hence, scalability. Therefore, scalable distributed file sharing system is made possible by the decentralization of server files within the intranet such that they do not reside only in one server.

However, this is the bane of the full implementation of Distributed File System in most institutions in Nigeria given that we use only a centralized and single server network [7].

2.3. Implementing Scalability in Distributed File Sharing System

As earlier stated about the closer study of this relevant technological solution, Distributed File System (DFS) shows that it presents a variety of challenges that cannot be ignored. And the key problem it presents is scalability (especially when a server is to be mounted or dismounted). Therefore, the extra factors to consider while implementing scalability include:

1) Number of servers: Recall that scalability implies that there are more than one server systems in the distributed environment, and no client should be down when a server within the network fails given that other servers are still up and files are being replicated into each of them. Hence, the number of servers assures and maintains the availability and reliability of an intranet.

2) Operating System: Among other network operating systems, LINUS is recommended considering its robustness and shield against viruses.

2.4. Characteristics of a Scalable Distributed File System

Scalable client/server system has the following characteristics

1) Clients are no more affected by back-end issues such as server failure.

2) The front-end application can be accessed on any computer system within the network; hence, defeating the problem of unwanted and unauthorized accessibility.

3) Security of network (or organizational) data is assured.

4) Clients will experience easy transportation of their files within the network environment.

The server engines are developed to communicate among the entire computers within the network; hence, offering and ensuring transparency.

3. Summary

With rapid increasing tasks/ workload(s) and workflow(s) in our present day establishments, and the challenges posed to the existing file sharing systems and their access control models by the internet, there is a need for a scalable file system. These days, big institutions desire to have an automated workflow that would help eliminate the tedious manual processes of communicating and sharing files/mails. Scalable distributed file sharing system is thus, a potential candidate. Among all the file systems, network file system (NFS) is the most proven technique. In this paper, each user/client within the network can access the organization's network system as though the files are resident on his local system. Also, two or more servers can be used in order to achieve scalability of the Distributed File System. Thus, the institutional workflow is not halted when a server goes down because there is a fall back server which keeps the network system/environment running.

This paper has thus far achieved its objective as stated which is to propose the implementation of "scalability" feature of distributed file sharing system, as a solution to the Distributed File System (DFS) problems earlier mentioned.

4. Conclusion

The transition from traditional centralized storage system and poor file system (network attached disks) to a scalable Distributed File System has been presented in this paper. The challenge of scalability, especially when a server is to be mounted or dismounted, and slow transportation of files can be eliminated with the introduction of a decentralized server-based system by the use of two or more servers that run synchronously. And when scalability is achieved, clients will no longer be affected by the back-end issues such as a server failure. The system will also guarantee an easier/quicker transportation of files among client systems.

5. Recommendation

It is highly recommended that organizations such, Nigerian tertiary institutions adopt and implement the use of two or more server nodes when building their intranet system in order to achieve a scalable distributed file sharing system which is a robust and more reliable networked environment for every 21st-century organization.

Conflicts of Interest

The authors declare no conflicts of interest regarding the publication of this paper.

References

[1] Wikipedia (2019) History of Information Technology. http://openbookproject.net/courses/intro2ict/history/history.html

[2] Rouse, M. (2005) File Sharing. https://searchmobilecomputing.techtarget.com/definition/file-sharing

[3] Wikipedia (2017) File Sharing. https://en.wikipedia.org/wiki/File_sharing

[4] Rouse, M. (2005) Distributed File System. https://searchwindowsserver.techtarget.com/definition/distributed-file-system-DFS

[5] Levy, E. and Silberschatz, A. (1990) Distributed File Systems: Concepts and Examples. *ACM Computing Surveys*, **22**, 321-374. https://doi.org/10.1145/98163.98169

[6] Bžoch, P. (2012) Distributed File System. http://www.google.com

[7] Depardon, B., Le Mahec, G. and Séguin, C. (2013) Analysis of Six Distributed File Systems. Research Report, 44 p.