A Practical Significance Distributed File System: An Analysis

Gajendra Sharma[1*], Gaurav Koirala[1]
[1]*Department of Computer Science and Engineering, Kathmandu University, Dhulikhel, Kavre, Nepal*
Corresponding Email[*]: gajendra.sharma@ku.edu.np

Abstract

The world's data collecting is growing and expanding. The infrastructure must be able to hold a large amount of data, which is becoming more crucial every day. Distributed File Systems could be used to accomplish this. For processing, storing, and analyzing massive amounts of unstructured data, the Distributed File System (DFS) is utilized. Google File System (GFS) and Hadoop Distributed File System are the most popular file systems (HDFS). We give a review of distributed file system design in this work. Scalability, availability, compatibility, extensibility, autonomy, and other factors are considered while comparing file systems.

Keywords

Distributed File System, Big Data, Design of DFS, Hadoop Distributed File System (HDFS), Google File System (GFS), Review of DFS

1. INTRODUCTION

Data is generated from a variety of sources, including business operations, transactions, social networking sites, web servers, and so on, and it can be structured or unstructured [1].Data sharing in distributed systems is already ubiquitous and becoming more so. In a distributed system, each user can be both a creator and a consumer of data. The Distributed File System (DFS) is a set of client and server services that enable an organization to organize numerous SMB file shares into a distributed file system using Microsoft Windows servers. By allowing shares in numerous different locations to be logically aggregated under one folder, it provides location transparency and redundancy to increase data availability in the face of failure or severe traffic. This type of system allows programs to store and access remote data in the same way that local files are stored and accessed, allowing users to access these files from any computer on the network. It is designed for batch processing, which means that data locations are accessible, allowing computation to move to where the data is stored while maintaining high bandwidth. DFS also provides capabilities like data sharing for numerous users, user mobility, and Location Transparency, which allows you to hide or not reveal any hint or tip about the file's location. The necessity for a Distributed File System arose from the disadvantage of the previous system, which needed users to know the physical addresses of all computers involved in the file sharing process.

When compared to a local file system, the requirements for a distributed file system are different. The requirements that must be considered when designing the Distributed File System are as follows [2]:

- The feature of fault tolerance must be well-implemented. One of the most

critical needs here is how quickly data can be restored following a failure.

- The size of the files kept in DFS will be enormous. The majority of the files are more than 1 GB. In DFS, handling these kinds of large files is critical. Some file systems will split files into chunks. The benefit of doing so is that the amount of data handled by a single operation is reduced from several GBs to a few MBs. However, it necessitates additional mapping procedures for each operation, which may result in a performance reduction.

- The majority of DFS files follow the write-once-read-many paradigms. As a result, many DFSs provide features that are optimized for file writers and readers [3]. Few of them also include effective functions for editing an existing file at any point in time. Some DFSs don't even have a feature that allows you to update an existing file.

- DFS relies heavily on metadata. Because most DFS systems accommodate millions of files, accessing every node directly is not an effective way to retrieve information on any given file. As a result, most DFS designate a single node as the central, which is responsible for maintaining the metadata of all files stored in the system. The metadata list will greatly speed up the retrieval of file information.

Nowadays, DFS enables the operation of massive data, large-scale computations, and transactions. The categories are based on the working logics and designs of the systems. Fault tolerance, replications, naming, synchronization, and design intent were used to make these classifications. The architecture of the Distributed File System can be understood from Figure 1.

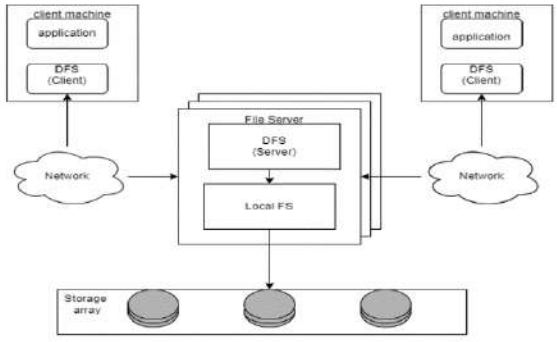

Figure 1: Distributed File System General Architecture

This paper is structured as follows. In Section 2, the background information required for the better understanding of DFSs presented in this paper is discussed. In this section, first the precise definition of the Distributed File System (DFS) is expressed. The most common Distributed File System architecture is explained, and in Section 3, the studies in design of DFS are presented. Finally, in Section 4, the general conclusion is discussed.

2. BACKGROUND

2.1 Design Goals of DFS

Early distributed file systems relied on a distant server to execute file system calls, which limited scalability and fault tolerance [4]. Modern approaches such as distributed hash tables, content-addressable storage, distributed

consensus algorithms, and erasure codes have considerably reduced such constraints. Two themes are emerging in view of projected exabyte-scale scientific data volumes [2]. First, distributed file systems' previously monolithic design is fragmented into services that provide a hierarchical namespace, data access, and distributed coordination individually. Second, the separation of storage and processing resources results in a storage architecture in which each compute node provides permanent storage. Google File System (GFS) and Hadoop Distributed File (HDFS) System are the most popular file systems.

In a number of ways, distributed file systems may strive for transparency. That is, they strive to be imperceptible to client programs, which see a system that resembles a local file system [2]. The distributed file system is in charge of locating files, transmitting data, and perhaps delivering the other features described below [5].

- Access transparency; Clients are ignorant that files are dispersed and can access them in the same way that local files are accessed.
- Location transparency;There is a consistent namespace that encompasses both local and distant files. The location of a file is not indicated by its name.
- Failure transparency; the client and client programs should operate correctly after a server failure.
- Replication transparency; to support scalability, we may wish to replicate files across multiple servers. Clients should be unaware of this.
- Heterogeneity; file service should be provided across different hardware and operating system platforms.

2.2 Issues in Distributed File System

When developing a distributed file system, there are a number of considerations to keep in mind. Transparency, flexibility, reliability, performance, scalability, and security are the several topics discussed in this section [6].

A. Transparency

The most important design issue is to hide from the users the fact that processes and resources are physically distributed across the network.

B. Flexibility

The easiest strategy to achieve flexibility is to decide whether each machine should use a monolithic kernel or a microkernel. Memory management, process management, and resource management are the three main functions of the kernel.

C. Reliability

Users prefer a distributed system with numerous processes because it protects them from single-processor system failures. As a result, in the event of a failure, a backup is available. The term "reliability" refers to the availability of data that is free of errors. All copies of a replication should be consistent.

D. Performance

It simply means that a program should run as though it were on a single CPU. Response time, throughput, system utilization, and the amount of network capacity consumed are the measures used to assess performance.

E. Scalability

Distributed systems are made to work with a small number of CPUs. It's possible that the distributed system will need to be expanded by adding more CPUs. There are limitations with centralized services, tables, and algorithms when supporting additional people or resources.

F. Security

Security consists of three main aspects namely,

1. Confidentiality which means protection against unauthorized access

2. Integrity, which implies protection of data against corruption

3. Availability, which means protection against failure and always being accessible

G. Fault Tolerance

In case a system has multiple servers' and if any Server breaks down, the other server takes up the load. This process is transparent to the user.

2.3 The Most Common DFS

Google engineers provided a solution to the problem in 2003, when the Internet giant faced scaling concerns and issues with storage media and the establishment of a distributed file system. The Google File System (GFS) refers to a collection of documents that have resulted in a much larger distributed file system application that is very similar to the GFS. The Apache Hadoop Distributed File System (HDFS) has been presented as a system file.

Google File Systems (GFS)

The Google File System (GFS) was first introduced in 2003 to suit Google's fast rising data processing demands. It's a scalable distributed file system designed for large-scale, data-intensive distributed applications. It has fault tolerance and gives great aggregate performance to a large number of clients while running on low-cost commodity hardware. GFS clusters are made up of hundreds, if not thousands, of storage units made from low-cost commodity hardware [7].

A. Goals

GFS shares many of the same goals as previous distributed file systems such as performance, reliability, and availability. Some of the design goals specific to GFS are as given below.

- Redundant storage of massive amounts of data on cheap and unreliable computers.
- It has to process huge numbers of requests
- GFS stores a huge number of files, totaling many terabytes of data.

B. Processes

GFS servers are Stateful. They have to maintain all the state information about the requests made by the clients.

C. Fault Tolerance

In GFS, all data is triple replicated. When a chunk server goes down, the master can always send data requests to the replicas until the node is up

and running again. In the event that the master fails, the system can easily select another node to build a metadata list by scanning all chunk servers and acting as master.

Hadoop Distributed File System (HDFS)

HDFS (Hadoop Distributed File System) is a fault-tolerant distributed file system designed to run on commodity hardware. HDFS is a file system that allows high-throughput access to application data and is well-suited to applications with huge data collections.

Hadoop includes a distributed file system (HDFS) that can store data across thousands of computers, as well as a way to spread work (Map/Reduce jobs) across those machines, allowing the work to be done close to the data [8]. The architecture of HDFS is master/slave. Large data sets are automatically divided into chunks that are maintained by multiple Hadoop nodes.

A. Goals

- As HDFS is designed for batch processing rather than interactive use by users, the emphasis is on high throughput of data access rather than low latency of data access.
- Detection of faults, quick and automatic recovery from them is a core architectural goal of HDFS.
- It should support tens of millions of files in a single instance.

B. Processes

HDFS servers are Stateful. They have to maintain all the state information about the requests made by the clients.

C. Fault Tolerance

One copy is placed on one node in the local rack, another on a different node in the local rack, and the final replica is placed on a separate node in a different rack, according to HDFS's placement rules. If the local rack's initial node fails, a request is forwarded to a different node in the same rack, where the replica is located. If that node fails as well, the request is sent to another node in a different rack. Fault tolerance is achieved in HDFS as a result of the use of replica placement policies.

3. STUDIES IN DESIGN OF DFS

There has been a lot of research into various designs. Only a few examples are mentioned in this article. This section focuses on the activities performed for unstructured data, small picture storage, and the benefits of these distributed file system designs. So, first, P2P and Client/Server technologies will be introduced in DFS design, followed by DFS designs based on HDFS and GFS. Table 1 outlines the many types of distributed file system designs and benefits relevant to the current study.

3.1 Related studies on P2P technology

Depending on the structure of the overlay maintained by participating peers, peer-to-peer networks are divided into two groups. Unstructured architectures, such as the pioneering Gnutella system, allow peers to create their own message routing paths [9]. To

access a resource, each peer will send requests to his virtual neighbors, effectively overloading the physical infrastructure (at least, this was the initial technique, which was eventually superseded by more "clever" routing algorithms).

DFS Peers are structured in both topologies, according to the DFS Architecture. A peer layout that is similar to that described in is discussed in. Because resource dependencies and links between participating nodes might be arbitrary, the DFS collaboration network resembles an unstructured peer-to-peer design. However, the overlay of namespace and storage aggregation and distribution is followed by a subset of requests. The location of DFS users and resources is managed using a Kademlia-based DHT shared by all peers in the DFS universe. Work on peer-to-peer content distribution networks (CDNs) and the Freenet censorship-free data delivery overlay is also relevant.

Distributed file systems are the forerunners of most modern peer-to-peer and distributed file sharing services, and as such have supplied the tools and mentality required to imagine contemporary global-scale data collaboration infrastructures from the start [4]. Frangipani and XFS are two of the most well-known early distributed file system ideas. Frangipani uses a two-layer method, separating file metadata from actual storage (a shared virtual disk).

3.2 Related studies on Client/Server technology for DFS Design

The majority of client/server applications fit into one of two categories: file server, or stateless, designs, and database styled transactional, or stateful, systems. Despite the fact that many client/server systems do not manage files or any other type of database, most of them have a design that is fairly similar to one of them [1]. Furthermore, there is a significant middle ground consisting of non-transactional stateful distributed architectures (including stateful file servers). These applications are sometimes referred to as "improved stateless" architectures [10].

For example, Microsoft's NTFS file system appears to be stateful to the user, but it is implemented as a "mostly stateless" system that uses event notification mechanisms to notify the client when events on the server that may be important to it occur; the client quickly rereads the changed data, and, with any luck, applications running on it will not notice the temporary inconsistency [11]. If one understands the fundamental ideas behind stateless system designs, a file system like this can be thought of as starting with a stateless approach and then cleverly adding mechanisms that hide many of the common issues found in stateless designs—an approach that gives the Microsoft system significant robustness. For example, the Web Services architecture invites one to follow a similar development path.

A study [11] presents a file system that keeps file replications in separate network areas. In the suggested system, they also construct a load detecting module. The module will monitor each file server's load and optimize overall system performance. A distributed file

system called KIV-DFS is presented by [12]. The Google file system is incompatible with mobile devices. The system is built on the client-server model. Users can work with data using the client module.

A distributed file system is presented in the work by [6]. The Name Server is in charge of data classification and storage allocation in this system. The Name Server was previously utilized to assist the Name Server's operations. A Data Server is a form of image to save that is placed beneath the Name Server. Data Servers are unrelated, and they have no idea what type of video is saved on the others. A fundamental technique to managing photos is to define the data and then use the descriptive information to carry out the operation. The image data is described using raw data, basic features, low-level features, and semantic features. Image data is shown using a tree structure. Raw data is the base node, and it refers to image data files. Low-level properties and key attributes of states are found in the middle child of the root node. Image data aspects such as color, texture, and geometry of images are examples of low-level features. Attributes such as name, kind, author, and creation time are all basic aspects. The leaf node, which expresses semantic properties such as a good writer, topic explanation, and low-level features [8].

3.3 Related studies in DFS based on Google File System

The Google File System is a scalable distributed file system created and implemented by Google for large distributed data-intensive applications. It has fault tolerance and gives great aggregate performance to a large number of clients while running on low-cost commodity hardware [9]. While it has many of the same goals as prior distributed file systems, this design was influenced by present and future observations of application workloads and technology environments, which differ significantly from previous file system assumptions. As a result, established design decisions have been reexamined, and radical new design points have been explored.

There are two replicas in the GFS: primary and secondary. The data chunk that a chunk server transmits to a client is known as a primary replica. On other chunk servers, secondary replicas act as backups. Which portions are primary or secondary is decided by the master server [12]. If a client modifies the data in a chunk, the master server notifies chunk servers with secondary replicas that they must copy the updated chunk from the primary chunk server in order to keep the chunk in its present state.

3.4 Related studies in DFS based on HDFS

Hadoop is a free and open-source software project [8]. Apache Hadoop has become a prominent corporate cloud-based technology applied as a whole ecosystem of services, thanks to a considerable technology investment supplied by Yahoo! Many multinational organizations, like Facebook, LinkedIn, Twitter, eBay, Samsung, J.P. Morgan, AOL, and others, have successfully implemented it [3]. The ecosystem includes a number of tools (Hive, Pig, HBase, Oozie, Zookeeper, Sqoop, Flume, Spark, Kafka, Impala) that enable businesses to store,

process, and analyze large amounts of data, provide real-time customer service, perform various types of optimizations, machine learning, ETL (extract, transform, load), and other operations.

HDFS is a distributed file system that handles files across the storage of a cluster. It's in charge of data replication and fragmentation. Depending on the file replication factor, large files are split into numerous blocks, each of which is replicated on multiple servers. This ensures that data is accessible even if one or more servers fail. The master-slave architecture is used by the distributed file system, which follows the "Write Once, Read Many" principle. It's a distributed "keyvalue" data storage system that's fault-tolerant and designed to manage huge files.

Clover, a NameNode cluster file system based on HDFS, is presented in the paper [9]. This file system takes advantage of two key features: an upgraded 2PC protocol that enables consistent metadata updates across many metadata servers, and a shared storage pool that provides reliable permanent metadata storage and facilitates distributed transaction operations [9]. Experiments demonstrate that by using quantized measurements, their system may provide superior metadata expandability ranging from 10% to 90% when each more server is added, while maintaining equivalent I/O performance Designed to work with big data files [6].

HatS [5]is a multi-tiered storage system based on the HDFS file system [7]. The DataNode architecture is a significant difference between hatS and HDFS. Each participating node in HDFS [6] maintains a single DataNode instance, which represents various storage devices, regardless of their features such as supported I/O rates and capacity. In hatS, on the other hand, each participating node hosts many DataNode instances, each of which represents a single type of storage device. Liu Jiang and colleagues [7] presented an optimization of a file system based on small files that delivers higher performance. A similar design with HDFS was presented by Farag Azzedin [10], although NameNode is distributed.

Article Name	Author (s)	Publish	Year	Pages	Advantages	Name of DFS
A High Performance and Low Cost Distributed File System	Tzong-Jye Liu, Chun-Yan Chung, Chia-Lin Lee	IEEE	2015	36-42	low cost, reliability, scalability, improve the overall system performance	NFS
Distributed File System with Online MultiMaster Replicas	Luboš Matějka, Jiří Šafařík, Ladislav Pešička	Second Eastern European Regional Conference on the Engineering of Computer Based Systems	2019	13-18	Scalability proper for mobile devices, Increase throughput, Increase the security of the data, improves the service availability	KIVDFS
ASDF: An Autonomous and Scalable Distributed File System	Chien-Ming, Wang, Chi-Chang Huang, Huan-Ming Liang	14th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing	2014	485-493	Scalability, Reliability, Compatibility, extensibility, autonomy	ASDF
A Kind Of Distributed File System Based On Massive Small Files Storage	DI LIU, SHT-JIE KUANG1	IEEE	2012	394-397	improve metadata management, high concurrency, storage efficiency, simple design, efficient design, storage capacity, scalability	QFS
The Hadoop Distributed File System	Konstantin Shvachko, Hairong Kuang, Sanjay Radia, Robert Chansler	IEEE	2010	1-10	Scalability, improves the overall availability, all of Goals in DFS design	HDFS

Distributed file system and classification for small images	Shaojian Zhuo, Xing Wu, Wu Zhang and Wanchun Dou	IEEE International Conference on Green Computing and Communication	2013	2231-2234	Reduce meta data, High Speed access	
hatS: A HeterogeneityAware Tiered Storage for Hadoop	Krish K.R., Ali Anwar, Ali R. Butt	18th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing	2018	502-511	Improve throughput , improve job completion time, improve the utilization of the storage device	Hats
Clover: A distributed file system of expandable metadata service derived from HDFS	Youwei Wang, Jiang Zhou, Can Ma, Weiping Wang, Dan Meng, Jason Kei	IEEE International Conference on Cluster Computing	2019	126-134	Expandability, bottleneck-less, no single point of failure	Clover
Towards a scalable HDFS architecture	F. Azzedin	IEEE International Conference on Collaboration Technologies and Systems (CTS	2017	155-161	Highly available, Widely scalable, fault tolerant,	
The optimization of HDFS based on small file	Liu Jiang, Bing Li, Meina Song	the 3rd IEEE International Conference on Broadband network and Multimedia Technology	2015	912-915	Increase the speed of reading, reduce the read request, reduce the write request received by the name node	

4. RESULT

Big data is a term for a collection of data that can be gathered, refined, managed, and processed using standard software in a reasonable amount of time. The definition of "size" in the context of big data is always evolving and expanding. As a result, a distributed file system should be able to handle and process such a massive amount of data. In Cloud and Grid computing, integrating enormous dispersed storage systems to provide large storage capacity is a critical and difficult task.

Various distributed file system designs were discussed in this article. A quick summary of how we used Distributed File Systems to achieve our goals using big data was offered. As a result, all DFSs are used to process, store, and analyze massive amounts of unstructured data. The Google File System exemplifies the characteristics required to run large-scale data processing workloads on commodity hardware. While certain design decisions are unique to our situation, many others may be applied to data processing tasks of equal scope and cost.

Although the speed of INetwork connectivity is critical for HDFS performance, it is not the only constraint. The data transfer speeds of hard drives are also crucial, especially in high-speed situations. Networks that serve a specific geographic area, such as a city or a metropolis. If you are uploading or writing a file to the HDFS. A data node, one of the replicas, hosts the client.will almost certainly be saved on the same data node.

REFERENCES

[1] A. B. Patel, M. Birla, and U. Nair, "Addressing big data problem using Hadoop and Map Reduce," 2012. doi: 10.1109/NUICONE.2012.6493198.

[2] T. J. Liu, C. Y. Chung, and C. L. Lee, "A high performance and low cost distributed file system," 2011. doi: 10.1109/ICSESS.2011.5982251.

[3] S. Ghemawat, H. Gobioff, and S. T. Leung, "The google file system," in *Operating Systems Review (ACM)*, 2003, vol. 37, no. 5. doi: 10.1145/1165389.945450.

[4] C. M. Wang, C. C. Huang, and H. M. Liang, "ASDF: An autonomous and scalable distributed file system," 2011. doi: 10.1109/CCGrid.2011.21.

[5] K. R. Krish, A. Anwar, and A. R. Butt, "HatS: A heterogeneity-aware tiered storage for hadoop," 2014. doi: 10.1109/CCGrid.2014.51.

[6] S. Zhuo, X. Wu, W. Zhang, and W. Dou, "Distributed file system and classification for small images," 2013. doi: 10.1109/GreenCom-iThings-CPSCom.2013.422.

[7] J. Liu, L. Bing, and S. Meina, "The optimization of HDFS based on small files," 2010. doi: 10.1109/ICBNMT.2010.5705223.

[8] K. Shvachko, H. Kuang, S. Radia, and R. Chansler, "The Hadoop distributed file system," 2010. doi: 10.1109/MSST.2010.5496972.

[9] Y. Wang, J. Zhou, C. Ma, W. Wang, D. Meng, and J. Kei, "Clover: A distributed file system of expandable metadata service derived from HDFS," 2012. doi: 10.1109/CLUSTER.2012.54.

[10] F. Azzedin, "Towards a scalable HDFS architecture," 2013. doi: 10.1109/CTS.2013.6567222.

[11] D. Liu and S. J. Kuang, "A kind of distributed file system based on massive small files storage," 2012. doi: 10.1109/ICWAMTIP.2012.6413521.

[12] L. Matějka, J. Šafařik, and L. Pešička, "Distributed file system with online multi-master replicas," 2011. doi: 10.1109/ECBS-EERC.2011.12.