Survey of Distributed File Systems: Concepts, Implementations, and Challenges

Kiet Tuan Pham, Sangjin Lee, Lan Anh Nguyen, Jeongyeup Paek, and Yongseok Son
Department of Computer Science and Engineering, Chung-Ang University

***Abstract*—Modern computing now relies heavily on Distributed File Systems (DFS), which make it possible to store, manage, and retrieve data across remote contexts in an effective manner. Therefore, it is essential to comprehend the DFS environment to optimize data management and storage solutions. We explore DFS in this thorough overview work, discussing its foundational ideas, well-known applications, related difficulties, and potential future research avenues.**

***Index Terms*—Distributed File Systems, cloud computing, data-intensive computing**

I. Introduction

The handling and storage of large volumes of information have become crucial to the success of organizations, research, and technical developments in today's era of data-driven computing. In order to meet the demands of dispersed computing environments, the need for high availability, and the issues provided by the exponential growth of data, DFS have become an essential solution.

The idea of a standard file system is expanded by a DFS to numerous networked nodes, most of the time dispersed over various distinct location, rather than just single machine. This extension makes it possible to efficiently organize, store, retrieve, and share data in situations when the volume of data is too great for a single system to manage or where data accessibility and redundancy are crucial.

The significance of DFS is underscored by their central role in modern computing ecosystems. In an era where data generation is incessant – from scientific simulations and sensor networks to social media interactions and business transactions – traditional storage models often fall short. DFS offer an innovative approach that addresses the challenges of scalability, fault tolerance, and data locality [1].

This survey paper aims to delve deeper into the realm of DFS. It will explore the foundational concepts, describe DFS taxanomy, discuss notable implementations, address challenges related to data security, consistency models, and performance optimization, and highlight emerging trends and future directions in the field.

II. Key Concepts of DFS

DFS introduce several fundamental concepts that underpin their functionality. They are: transparency, scalability, performance, redundancy, consistency, security and fault tolerance. [1]

A. Transparency

DFS provide location, access, migration, relocation and replication transparency, making it possible for users and programs to access files without being aware of where they are physically stored. This improves user experience by making data access and administration simpler.

B. Scalability and Expansion

Scalability is one of DFS's main benefits. By adding additional storage nodes, these systems may expand horizontally and easily handle increasing data volumes, which is suitable for big servers.

DFS are made to enable organisations to easily increase storage capacity by introducing additional nodes as their data needs increase. With this scalability, the system can grow to handle rising data quantities without compromising performance.

C. Data Replication and Redundancy

DFS employs data replication techniques to ensure fault tolerance and data availability. Replicating data across multiple nodes enhances reliability and reduces the risk of data loss.

DFS often uses methods like data replication or erasure coding to reduce the risk of data loss caused by network or hardware problems. These processes guarantee a high level of data availability and dependability.

D. Consistency and Coherency

It is difficult to maintain data consistency in a distributed environment. Therefore, DFS use a number of protocols and techniques to guarantee that data copies stay coherent and preserve data integrity.

The majority of DFS employ checksum to verify consistency by evaluating the data after transmission over communication network. Additionally, caching and replication are also crucial to DFS, especially when they are intended to run over wide-area networks. There are a number of different methods, including server-side replication and client-side caching. Data object replication and metadata replication are the two forms of data that need be taken into account for replication [4]. All DFS include a means to guarantee the availability and recoverability of metadata and metadata, such as a backup metadata server and a metadata snapshot with transaction logs.

E. Security

Some of the most important security difficulties in DFS that need to be taken into account are issues with access control and authentication. Most DFS include security using contemporary security methods including privacy, authentication, and authorisation. In contrast, some DFS only rely on the mutual confidence between all nodes and clients [2].

F. Fault Tolerance

On object data, there are two methods for fault tolerance: **failure as exception** and **failure as norm**. Systems that use **failure as exception** will isolate the failing node or restore the system to its previous state of regular operation. In contrast, **failure as norm** systems use all types of data replication and rereplication whenever the replication ratio becomes dangerous. [1]

III. Taxonomy Of DFS

Numerous elements influence the DFS to include the most suited and appropriate file system [1]. Here is the taxonomy list:

A. Architecture

There are 5 types of DFS with different architectures [6]:

- **Client-server**: A uniform view of the server's local file system is offered by this kind of server. Clients are able to access the files kept on a server, enabling a heterogeneous group of processes running on various computers and operating systems to share a single file system. This method has the benefit of being mostly independent of local file systems. [1]
- **Cluster-based**: Cluster-based server is made up of a single master and several chunk servers. The benefit is that it is straightforward and enables a single master to manage a few hundred chunk servers. Three key architectural aspects of the Cluster-based DFS are often taken into account during design: decoupled metadata and data, reliable autonomous distributed object storage, and dynamic distributed metadata management. [1]
- **Symmetric**: Peer-to-peer technology is the foundation of symmetric file systems. It combines a key-based lookup technique with a distributed hash table-based approach for distributing data. All nodes in a symmetric file system can grasp the disk structures since the clients also host the metadata manager code.
- **Asymmetric**: In asymmetric file system, the file system and the accompanying disk structures are maintained by one or more specialized metadata managers.
- **Parallel**: Data blocks in parallel file systems are striped over several storage devices on numerous storage servers. All nodes can access the same files simultaneously thanks to the provision of support for parallel programs, enabling concurrent read and write capabilities. This crucial functionality is supported by the majority of the existing DFS.

B. Processes

Most of the DFS enable stateful processes because stateless design is challenging to implement. However, with a stateless design, clients can fail and restart without interfering with the operation of the system as a whole.

C. Communication

Since the DFS make the system independent from underlying operating systems, networks, and transport protocols, most of them employ the Remote Procedure Call (RPC) mode of communication. TCP and UDP are the two communication protocols to take into account in the RPC technique. The majority of DFS use TCP. UDP, however, is also taken into account for enhancing performance in a number of DFS.

D. Naming

Two widely used strategies are:

- **Central metadata server**: Decopuling metadata and data increases file namespace performance and solves the synchronization issue. However, the central metadata server can become a performance bottleneck, particularly in large-scale and high-concurrency scenarios. If the central server fails or experiences issues, it can impact the entire file system.
- **Distributed metadata server**: Distributed metadata server can improve scalability, reduce single points of failure, and enhance performance by allowing parallel access to metadata across multiple nodes but ensuring data integrity and avoiding conflicts become important challenges.

E. Synchronization

Applications deployed on the DFS are designed with a variety of locking mechanisms depending on their intended use. The **Write-once-read-many** access strategy is necessary for major usages. However, there are applications that demand the **Multiple-producer/single-consumer** access paradigm, such as search engines.

IV. Notable DFS Implementations

Several DFS have gained prominence due to their innovative designs and successful deployments provided in Table I:

- **Google File System (GFS)**: The Google File System introduced the concept of chunking large files into fixed-size blocks, enabling efficient storage and retrieval. GFS's focus on high throughput and fault tolerance influenced subsequent DFS designs. [7]
- **Lustre**: Lustre uses remote links to connect a comparatively few huge subtrees. The administrator can construct a new subdirectory as a remote directory on a separate metadata node from its parent. Then, in the new location, all new files and folders under that new subfolder are created. [2]
- **Hadoop DFS (HDFS)**: HDFS, a cornerstone of the Hadoop ecosystem, is designed for storing and processing massive datasets. Its master-slave architecture and data

TABLE I
NOTABLE DISTRIBUTED FILE SYSTEM IMPLEMENTATIONS

FS Name	Release year	Architecture	Communication	Naming	Synchronization	Consistency and Replication
GFS	2003	Asymmetric	RPC/TCP	Central metadata server	Write-once-readmany, Multiple-producer/ single-consumer, give locks on objects to clients, using leases	Server-side replication, Asynchronous replication, checksum, relax consistency
Lustre	2003	Asymmetric	Network independence	Central metadata server	Hybrid locking mechanism, using leases	Server side replication, Only metadata replication, Client side caching, checksum
HDFS	2006	Asymmetric	RPC/TCP & UDP	Central metadata server	Write-once-readmany, give locks on objects to clients, using leases	Server side replication, Asynchronous replication, checksum
iRODS	2006	Asymmetric	TCP	Central metadata server	Write-once-readmany, Multiple-producer/ single-consumer, give locks on objects to clients	Server side replication, Only metadata replication, Client side caching, checksum
Ceph	2007	Symmetric	RPC/TCP & UDP	Distributed metadata server	Multiple-producer/ multiple-consumer, locks on object	Server-side replication, Asynchronous replication, checksum, relax consistency
IPFS	2015	Symmetric	RPC/TCP	Distributed metadata server	Multiple-producer/ multiple-consumer	Checksum, eventually consistency
JuiceFS	2017	Symmetric	RPC/TCP	Distributed metadata server	Multiple-producer/ multiple-consumer	Server-side replication, Asynchronous replication, checksum, eventually consistency

replication mechanisms contribute to its reliability and fault tolerance. [4]

- **Ceph**: Ceph offers a unified storage solution encompassing object, block, and file storage. Its distributed architecture and scalability make it suitable for a wide range of applications, from cloud environments to high-performance computing. [2]
- **Integrated Rule-Oriented Data System (iRODS)**: iRODS operates as a middleware layer that abstracts the underlying storage systems and provides a consistent and rule-based framework for managing data. It allows users to define data management policies and automate data workflows based on these rules. [5]
- **InterPlanetary File System (IPFS)**: IPFS is based on a content-addressable storage model, where each piece of content is assigned a unique hash based on its content. This hash becomes its address on the network, which results in content can be retrieved by referencing its hash rather than a specific location, allowing for efficient distribution and retrieval of data.

V. CHALLENGES AND FUTURE DIRECTIONS

DFS face several challenges that drive ongoing research and development:

- **Data Security and Privacy:** Data security in DFS is still a top priority. Researchers are investigating safe data exchange protocols, access control systems, and encryption technologies.
- **Consistency Models:** Maintaining strong data consistency without sacrificing performance is a challenge. Research is still ongoing to create innovative consistency models that balance system performance with data integrity.
- **Performance Optimization:** Continuous research is conducted in the areas of improving read and write speed, cutting latency, and optimizing data placement. Data placement regulations, load balancing algorithms, and caching techniques are all intensively investigated.

ACKNOWLEDGMENT

This work was supported by the National Research Foundation of Korea (NRF) grant funded by the Korea government (MSIT), KIAT grant funded by the Korea Government (MOTIE), and Korea Institute of Science and Technology Information (KISTI). (No. NRF-2021R1C1C1010861, NRF-2022R1A4A5034130, KIAT-P0012724) (Corresponding Author: Yongseok Son).

REFERENCES

[1] T. D. Thanh, S. Mohan, E. Choi, S. Kim, and P. Kim, "A Taxonomy and Survey on Distributed File Systems," in 2008 Fourth International Conference on Networked Computing and Advanced Information Management, Sep. 2008, pp. 144–149. doi: 10.1109/NCM.2008.162.

[2] B. Depardon, G. L. Mahec, and C. Séguin, "Analysis of Six Distributed File Systems".

[3] "Comparative Analysis of Major DFS Architectures: GFS vs. Tectonic vs. JuiceFS," InfoQ.

[4] L. S. Rani, K. Sudhakar, and S. V. Kumar, "Distributed File Systems: A Survey," vol. 5, 2014.

[5] K. T. Pham, S. Lee, S. Cho, S. Kim, and Y. Son, "A Survey on Data Management using Integrated Rule-Oriented Data System," in 2022 13th International Conference on Information and Communication Technology Convergence (ICTC), Oct. 2022, pp. 1116–1118. doi: 10.1109/ICTC55196.2022.9953028.

[6] Y. Zhou, "Large Scale Distributed File Systems Survey".

[7] P. Macko and J. Hennessey, "Survey of Distributed File Systems Design Choices," ACM Trans. Storage, vol. 18, no. 1, pp. 1–34, Feb. 2022, doi: 10.1145/3465405.