A survey on Distributed File Systems

Zhanhao Liu
Department of Computer Science and Engineering
The Chinese University of Hong Kong
Email: zhliu6@cse.cuhk.edu.hk

Abstract—**The scale of data and files is increasing dramatically nowadays. Distributed File System (DFS) had been introduced since the 1970s to deal with the massive data storage problem. An ideal DFS can provide data scalability, fault tolerance, and high concurrency through partitioning and replication of data on many nodes. Additionally, the issues of file locality and availability are also important. Last but not least, the use of commodity devices is significant for cost efficiency and practical value of a DFS. Many DFS's have been proposed over the years and the techniques that they used to distribute and share files among different nodes varies, Client-Server Architectures, Cluster-Based Distributed File System, Symmetric Architecture, and so on. [1] In this paper, we will survey some representative DFSs from 1997 to 2011 in their design principles as well as performance in terms of scalability and fault tolerance. The systems surveyed are Frangipani [2], Google File System [3], Panasas [4] [5], Ceph [6], and TidyFS [7].**

1. Introduction

As the rapid development of big data and cloud computing area, the scale of data and files that people generates every day is increasing exponentially. Facebook has over one billion users and generating 30 petabytes of data every day, Twitter has 650 million users who tweet 500 million tweets a day, and Youtube has more than 100 hours of video being uploading every minute. How to store these data efficiently and reliably is becoming a hot topic in both industries as well as academic.

The traditional approach of storage–storing file and data in a single computer or server, no longer works in nowadays usage scenario. On the one hand, a single server can only store a limited amount of data, which can't cope with the demand of today's big data storage scenario. On the other hand, a stand-alone storage is a disaster when it comes to machine failure, you may risk to lost all your valuable data.

In order to make people store the massive amount of data efficiently (fast write), access their large file easily (fast read), as well as having their files available even if one computer crashes (fault tolerance), Distributed File System (DFS) had been introduced date to 1970s. DFS provides many advantages such as ease of data sharing, the reliability of file storage, resources management, and accessibility, large data storage capacity, etc. Many of today's high-performance computing frameworks like Hadoop and Spark, are heavily rely on these DFSs. In this paper, we study 5 representative DFSs (Frangipani [2], Google File System [3], Panasas [4] [5], Ceph [6], and TidyFS [7].) on their design philosophy, scalability, fault tolerance as well as their innovation points.

This paper is organized as follows. Section 1 describes the usage requirements of the Distributed File System. Section 2 is detailed analysis and comments of the five DFSs studied. We then compare the DFSs in Section 3 and Section 4 is the take away that we learn from these representative DFSs.

2. Background

The emergence of massive data and high performance distributed computing technology require distributed file system for better performance in generally four perspectives:

2.1. Reliability

The bottom-most requirement for a distributed file system is reliability. On one hand, the system needs to ensure that there is no error during the data read/write process from/to the DFS. On the other hand, it needs to make sure the data will not be lost due to all kinds of reason (node failures, power failures, human factors) in the heterogeneous cluster environments. Hardware failures are the norm rather than the exception in a large-scale data center. A good DFS should ensure that highly concurrent data access/update (which is also a norm in a large-scale data center) will not bring any damage to the data integrity, and be able to recover from hardware failure quickly.

2.2. Availability

A good distributed file system should be able to maintain a 24/7 service. During the maintenance of the server, power down, or hardware failure, some nodes in the clusters may be out of service. To ensure that data continues to be available in the presence of failures, the DFS needs to use some strategies like data redundancy and cross-rack storage to provide the data availability.

2.3. Performance

Another important issue for a DFS is its performance. One view to measure a DFS's performance is by its read/write throughput in sequential and random workloads. As DFS normally store large file into different nodes which are connected by the network, so the network delay needs to be considered seriously while reading/writing large file from/to multiple nodes.

2.4. Scalability

The scalability of a DFS can be evaluated by the increase of the storage capacity and throughput when nodes are dynamically and continuously added to the system. A typical DFS deployed in the resource-harvesting data-centers contains thousands of nodes, it should be able to work in a large-scale environment without degrading its file access performance as well as data availability. One important problem in achieving scalability is the need for decentralization.

3. Case Studies

In this section, we analyze and comment the five representative DFSs that we surveyed, focusing on their design philosophy, scalability, fault tolerance as well as their innovation points.

3.1. Frangipani

Frangipani is a scalable distributed file system that with simple design and easy administration, it is developed by researchers from Digital Equipment Corporation. Its main goal is to cope with the laborious file system administration problem.

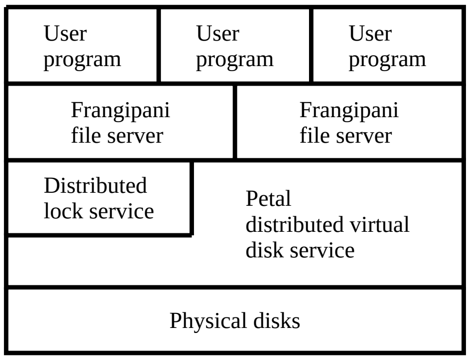

Figure 1. The layering of Frangipani [2]

3.1.1. System design. Base on the distributed shared virtual disk Petal [8], Frangipani adopts a novel two-layer structure. The lower layer is Petal, which provides an automatically managed virtual disks that can be accessed globally. In the upper layer, multiple nodes run the same file system code on top of Petal, which provides names, directories, and files. [2] In Frangipani, each client is also a server for the whole file system, participates in the management of the file system, and can access the Petal virtual disk equally and achieve synchronous access control via distributed lock service.

3.1.2. Scalability. The two-layer structure of Frangipani brings great scalability. In the file system layer, one can add storage devices dynamically without stopping the current operation or changing any machines' configuration. In the bottom layer, the Petal virtual disk separates the storage into block-level storage as well as the file system to achieve high scalability. In addition, the lock service of the system is also fully distributed for scalability concern.

3.1.3. Fault tolerance. Frangipani uses write-ahead redo logging of metadata and *Lease* [9] mechanism of the distributed lock service to achieve fault tolerance. By write-ahead logging, it means that when the le server of Frangipani want to update the metadata, it will first create a record describing the update and then appends it to the logging file. These log records are periodically written to Petal and can be used to recover file from failure. Also, the distributed lock service deals with client failures with *Lease*. A client first will obtain a lease and needs to renew it before some expiration time, otherwise, the server will consider it as a failure.

With this guarantee, Frangipani can automatically recover from server failures and continue to interact with the survived servers.

3.1.4. Potential issues.

- *Redundant logging*: Both the Petal virtual disk (bottom layer) and the Frangipani file system (upper layer) have their own logging, some of the may be redundant.
- *Whole-file locking*: Frangipani lock the entire files and directories rather than individual blocks when a writer wants to modify a file, this may introduce many lock revocation requests when there are many writers request to access the same file.

3.2. Google File System

To meet the rapidly growing data processing demands, Google introduced the Google File System (GFS) [3] in 2003. Inspired by Google File System, the open source Hadoop Distributed File System (HDFS) [10] is now the most commonly used DFS in industry.

3.2.1. Assumption. Apart from the normal goals (reliability, availability, good performance, and scalability) of DFSs, GFS is specially optimized for Googles core data storage and usage needs. The design of the GFS is guided by the following six assumptions:

- The system is built on many inexpensive commodity components that often fail.
- The system stores a modest number of large files.
- The workloads primarily consist of two kinds of reads: large streaming reads & small random reads.
- The workloads also have many large, sequential writes that append data to files.
- The system must eciently implement well-dened semantics for multiple clients that concurrently append to the same le.
- High sustained bandwidth is more important than low latency.

3.2.2. System design. The GFS is organized into clusters of nodes. Each GFS cluster consists of a single master, multiple chunkservers and accessed by multiple clients. The master node maintains all metadata of the le system, controls system-wide activities, and periodically communicates with chunkservers in *heartbeat*. Each file in GFS is divided into xed-size chunks (by default 64MB), with each chunk replicated (by default 3 replicas) and distributed over the chunkservers to increase the reliability of the system. Each chunk is assigned an immutable and globally unique 64 bit *chunk handle* for identification by the master during creation.

3.2.3. Scalability. There are over 1.5 billion websites on the world wide web now and Google handles 3.5 billion searches per day, thus Google File System needs to be scalable and available all the time. With the single master, multiple chunkservers architectures, the GFS can be easily expanded by adding more chunkservers into the clusters without significantly modifying the master server. Also, the master of GFS only tells the client which chunkserver that it needs to contact and does not involve in the read/write processes. This design can protect the master from being a bottleneck when the data and users increase.

3.2.4. Fault tolerance. Since each cluster in Google may have thousands of disks on hundreds of nodes, component failures are the norm rather than the exception in this scenario. To maintain a high availability as well as data integrity, GFS uses multiple techniques to deal with the failures:

- *Chunk Replication*: Each chunk of GFS in replicated on multiple chunkserves on different racks. Also, techniques parity or erasure codes are also applied to support the read-only storage requirements.
- *Master Replication*: As we state in Section 3.2.2, each GFS cluster has only one master. Thus, the master state of a cluster is replicated across multiple machines via operation log and checkpoints for reliability.
- *Checksumming*: The chunkserver of GFS uses checksumming to detect data corruption. When corruption detected, the master will create a new uncorrupted replica to replace the corrupted one.

3.2.5. Flaws.

- *Small files may become hotspots*: When a small file (single-chunk) is accessed by multiple clients simultaneously, it will become a hotspot and overload the chunkserver.
- *Single point of Failure*: Each GFS has only one master, if it fails, it takes time to recover from master replication and the file system may go down during the recovery time.

3.3. Panasas

The Panasas Parallel File System [4] [5] is the earliest object-based file systems in the industry. It's first developed in the PDL Lab of CMU and then turns into the Panasas Inc. focused on the high-performance data storage solutions area. The peculiarity of the Panasas system is its use of per-file, client-driven RAID, which can increase the system's security.

3.3.1. System design. The heart of the Panasas Storage Cluster is a decoupling of the datapath (i.e., read, write) from the control path (i.e., metadata), allowing the clients to directly communicate with each other. The main storage devices of Panasas are network-attached Object Storage Clusters (OSDs). Clients communicate with the OSDs directly to access the file data. Last, to achieve data redundancy and increase system's I/O throughput, the Panasas storage system strips the data of a file system across multiple objects.

3.3.2. Scalability. Traditional clustered NFS systems have limited scalability because they use a single access point to deal with clients' requests for accessing data. The Panasas storage system resolve this bottleneck by separating the computationally expensive metadata management from the bandwidth intensive datapath. Also, Panasas achieve a scalable bandwidth by striping data across multiple OSDs.

3.4. Ceph

Ceph [6] is an open source distributed file system designed to support object-based storage on a single distributed computer cluster, with excellent performance, reliability, and scalability.

3.4.1. System design. The Ceph file system can be generally divided into three components: client, Metadata Server (MDS), and the Object Storage Cluster (OSD). The client exposes a near-POSIX file system interface to a host or process and performs file I/O by communicating directly with OSDs, which store all the file and metadata of the file system. MDSs manage the namespace (file names and directories) and interact with the clients to perform metadata operations (open, rename). On the bottom layer of Ceph, it uses Reliable Autonomic Distributed Object StorageRADOSto achieve linear scaling by objects replications, cluster expansion, failure detection and recovery to OSDs.

TABLE 1. COMPARISION OF THE FIVE DFSS

	Frangipani	GFS	Panasas	Ceph	TidyFS
Architecture	Two-layer	Object-based	Object-based	Decentralized	Blob Store
Naming	Central metadata server	Central metadata server	Central metadata server	Distributed metadata	Central metadata server
Fault Tolerance	Redo logging, Leases	Heartbeat, Replication	Cache	Heartbeat, Monitor	Replication

3.4.2. Scalability. Ceph considers scalability from multiple dimensions, such as storage capacity, system throughput, and read/write performance. Like Panasas, Ceph decouples the data path and metadata management module. Traditional storage system maintains a centralized storage gateway (like the master is GFS) to act as a single point of contact for requests from clients. This single point of contact may impose a limit on system's scalability and introduce a single point of failure. Ceph finds its own way to use an algorithm called CRUSH [11] to efficiently compute information about object location, instead of keeping distribute object lists in its metadata cluster to highly improve its scalability.

3.4.3. Fault Tolerance. For fault tolerance, Ceph uses similar approaches like GFS, OSDs exchange heartbeats periodically to detected node failures. Additionally, Ceph supports a cluster of monitors to collect failure reports centrally.

3.4.4. Flaws.

- *Strong consistency control bad for large cluster*: RADOS use a strong consistency control (read-one write-all), this is good for a small file system that with much more read requests than write. However, in the case of a large cluster of nodes, this consistency control may greatly hurt the write performance.

3.5. TidyFS

TidyFS [7] is a simple and small distributed le system that provides the necessary abstractions for recent years' data-intensive parallel computations on clusters.

3.5.1. System design. There are mainly three components of the TidyFS storage system: the Metadata server, Node Service, and TidyFS Explorer. Like in GFS and HDFS, TidyFS use a central Metadata Server to store the mapping of streams to parts, machine state, mapping of parts to storage machine and data path, etc. On each storage machine, a node service is run on it to provide housekeeping services like garbage collection, replication, and validation. As for the TidyFS Explorer, it's a graphical user interface for the distributed le system, which allows the users to monitor the state of the system as well as interact with it.

3.5.2. Fault Tolerance. TidyFS developed its own tool called **Watchdog** to detect error conditions (such as replication failure) and alert conditions and then report to the administrator for manual correction. Just like GFS and HDFS, TidyFS replicate its data parts on multiple nodes to support fault-tolerance. When failures happen, the fault-tolerance module will re-execute the failed or slow processes elsewhere.

3.5.3. Issues with native interfaces. Differ from GFS and HDFS, TidyFS allow clients to access its file via native interfaces (e.g., NTFS or SQL Server). By doing this, it can allow the applications to use the most suitable parts access patterns to perform I/O. Also, this can avoid extra indirection layer between the client and the file system, which can maximize the clients' I/O performance. However, this design also brings some issues.

- *Lack of automatic eager replication*: To support the native interface access, TidyFS does not allow automatic eager replication except for optional eager replication in the data ingress case. The authors are happy with the tradeoff because their use cases are small file system deployment, so it's not common to have data lost before replication has completed.
- *Lack of control over part sizes*: Lack of control over data part size may result in some parts much larger than other ones. This can cause problems with the simple replication policy of TidyFS. Sometimes a defragmentation is needed.

4. Analysis

As the size of file and usage scenario evolves, the design of DFS evolves to accommodate these changes too. In the five DFSs that we surveyed, the oldest one is Frangipani that based on the distributed shared virtual disk Petal and a novel two-layer structure. Thought the Frangipani system may not be able to serve for today's big-data storage requirement, but some of its design philosophy, such as decentralization, failure recovery, distributed lock, have a great impact on subsequent DFS's design. (such as GFS)

As for the GFS, it shows us how the "biggest data center" in the world deal with data storage. In the trade-off between strong consistency and high performance, GFS choose higher throughput performance. The open-source project HDFS that strongly influenced by GFS, is still the most used distributed file system nowadays.

Panasas and Ceph are both parallel file system on top of object-based storage. The concept of decoupling of storage control from datapath operations in this two systems enables them to achieve a high scalability without a single access point.

To accommodate the prototypical workloads (high-throughput, write-once, sequential I/O) of today's parallel computing frameworks, TidyFS is developed by Microsoft for their community needs. Differ from other similar systems like GFS and HDFS, TidyFS is much simpler and allow clients to access the stored data using native interfaces.

5. Conclusion

Distributed file system has become a widely-used form of shared permanent storage by supercomputers, clusters, and data-centers. As the data dimension and computation power increase day by day, people need DFSs with high performance, scalability, reliability, and availability, to store the large amount of data. Besides performance, the cost of per unit storage, use of commodity devices to build the system, and the ability to handle specific kind of workload (like TidyFS is specially designed for parallel computing workload) are becoming hot topics in the area of DFS.

References

[1] T. D. Thanh, S. Mohan, E. Choi, S. Kim, and P. Kim, "A taxonomy and survey on distributed file systems," in *Networked Computing and Advanced Information Management, 2008. NCM'08. Fourth International Conference on*, vol. 1. IEEE, 2008, pp. 144–149.

[2] C. A. Thekkath, T. Mann, and E. K. Lee, "Frangipani: A scalable distributed file system," *SIGOPS Oper. Syst. Rev.*, vol. 31, no. 5, pp. 224–237, Oct. 1997. [Online]. Available: http://doi.acm.org/10.1145/269005.266694

[3] S. Ghemawat, H. Gobioff, and S.-T. Leung, "The google file system," *SIGOPS Oper. Syst. Rev.*, vol. 37, no. 5, pp. 29–43, Oct. 2003. [Online]. Available: http://doi.acm.org/10.1145/1165389.945450

[4] B. Welch, M. Unangst, Z. Abbasi, G. A. Gibson, B. Mueller, J. Small, J. Zelenka, and B. Zhou, "Scalable performance of the panasas parallel file system." in *FAST*, vol. 8, 2008, pp. 1–17.

[5] D. Nagle, D. Serenyi, and A. Matthews, "The panasas activescale storage cluster: Delivering scalable high bandwidth storage," in *Proceedings of the 2004 ACM/IEEE Conference on Supercomputing*, ser. SC '04. Washington, DC, USA: IEEE Computer Society, 2004, pp. 53–. [Online]. Available: https://doi.org/10.1109/SC.2004.57

[6] S. A. Weil, S. A. Brandt, E. L. Miller, D. D. E. Long, and C. Maltzahn, "Ceph: A scalable, high-performance distributed file system," in *Proceedings of the 7th Symposium on Operating Systems Design and Implementation*, ser. OSDI '06. Berkeley, CA, USA: USENIX Association, 2006, pp. 307–320. [Online]. Available: http://dl.acm.org/citation.cfm?id=1298455.1298485

[7] D. Fetterly, M. Haridasan, M. Isard, and S. Sundararaman, "Tidyfs: A simple and small distributed file system." in *USENIX annual technical conference*, 2011, pp. 34–34.

[8] E. K. Lee and C. A. Thekkath, "Petal: Distributed virtual disks," *SIGOPS Oper. Syst. Rev.*, vol. 30, no. 5, pp. 84–92, Sep. 1996. [Online]. Available: http://doi.acm.org/10.1145/248208.237157

[9] C. Gray and D. Cheriton, "Leases: An efficient fault-tolerant mechanism for distributed file cache consistency," *SIGOPS Oper. Syst. Rev.*, vol. 23, no. 5, pp. 202–210, Nov. 1989. [Online]. Available: http://doi.acm.org/10.1145/74851.74870

[10] K. Shvachko, H. Kuang, S. Radia, and R. Chansler, "The hadoop distributed file system," in *Proceedings of the 2010 IEEE 26th Symposium on Mass Storage Systems and Technologies (MSST)*, ser. MSST '10. Washington, DC, USA: IEEE Computer Society, 2010, pp. 1–10. [Online]. Available: http://dx.doi.org/10.1109/MSST.2010.5496972

[11] S. A. Weil, S. A. Brandt, E. L. Miller, and C. Maltzahn, "Crush: Controlled, scalable, decentralized placement of replicated data," in *Proceedings of the 2006 ACM/IEEE conference on Supercomputing*. ACM, 2006, p. 122.