A Taxonomy and Survey on Distributed File Systems

Tran Doan Thanh[1], Subaji Mohan[1], Eunmi Choi[1]*, SangBum Kim[2], Pilsung Kim[2]
[1]School of Business IT, Kookmin University, Seoul, Korea
thanhtd@kookmin.ac.kr, subajimohan@yahoo.co.in, emchoi@kookmin.ac.kr
[2]SK Telecom Convergence and Internet R&D Center / SK T-Tower, Seoul, Korea
amzang@sktelecom.com, pskim11@sktelecom.com

Abstract

Applications that process large volumes of data (such as, search engines, grid computing applications, data mining applications, etc.) require a backend infrastructure for storing data. The distributed file system is the central component for storing data infrastructure. There have been many projects focused on network computing that have designed and implemented distributed file systems with a variety of architectures and functionalities. In this paper, we develop a comprehensive taxonomy for describing distributed file system architectures and use this taxonomy to survey existing distributed file system implementations in very large-scale network computing systems such as Grids, Search Engines, etc. We use the taxonomy and the survey results to identify architectural approaches that have not been fully explored in the distributed file system research.

1. Introduction

The Distributed File System (DFS) is used to build a hierarchical view of multiple file servers and shares on the network. Instead of having to think of a specific machine name for each set of files, the user will only have to remember one name; which will be the 'key' to a list of shares found on multiple servers on the network. *Permanent Storage* is a fundamental abstraction in computing. A permanent storage consists of a named set of objects that (1) come into existence by explicit creation, (2) are immune to temporary failures of the system, and (3) persist until explicitly destroyed. The naming structure, the characteristics of the objects, and the set of operations associated with them characterize a specific refinement of the basic abstraction. A file system is one such refinement.

A DFS is a file system that supports the sharing of files in the form of persistent storage over a set of network connected nodes [4]. Many DFS's have been developed over the years and almost two decades of research have not succeeded in producing a fully-featured DFS[1, 2, 3].

Multiple users who are physically dispersed in a network of autonomous computers share in the use of a common file system. A useful way to view such a system is to think of it as a *distributed implementation* of the timesharing file system abstraction. The challenge is in realizing this abstraction in an efficient, secure and robust manner. In addition, the issues of *file location* and *availability* assume significance. One way of increasing the availability of files within a DFS is by using the replication of files. Most of the replication techniques can be divided into two main categories such as optimistic replication and pessimistic replication [5].

Another major bottleneck in the performance of DFS is the dramatic improvements in the processor speeds. To overcome this limitation DFS uses caches at various points [7] and these caches can be positioned at either the file server or at the client [6]. To provide a consistent view of the data seen by all clients in a DFS and reliability in the case of failures, write operations are allowed to complete only after the data has been committed to stable storage. Therefore, the dominant loads on the file server are due to writes. Thus allowing write-backs from client can reduce this write-load on the server [7, 8].

The ability to use commodity devices for easily and *economically* scale-up is now very important in DFS's because of the demand of large-scale distributed applications. This includes the *incremental scalability* which is the ability to add more devices to scale up the system in incremental fashion.

The systems surveyed are Google File System (GFS) [15], Lustre [16], Kosmos File System [17], Hadoop Distributed File System (Hadoop) [14], Panasas [18], Parallel Virtual File System (PVFS2) [19], and Redhat Global File System (RGFS) [20]. Requirements for DFS were described and an abstract functional model developed. The requirements and

* Corresponding author: Eunmi Choi (emchoi@kookmin.ac.kr). This work is supported by SKT research project fund in 2008.

978-0-7695-3322-3/08 $25.00 © 2008 IEEE
DOI 10.1109/NCM.2008.162

model were used to develop the taxonomy. This helped in identifying some of the key DFS approaches and issues that are yet to be explored and we expect such unexplored issues as topics of future research. A comprehensive bibliography forms the importance of the paper.

The structure of the paper is as follows. Sections 2 cover Background information about Distributed File System. In section 3, Taxonomy of Distributed File System is reviewed in detail. In Section 4 overview of different Distributed File Systems and comparison between them are shown and in Section 5, Findings related to DFS are presented and in Section 6 Discussion and Conclusion of the paper is outlined

2. Background

This review was started with the basic abstraction of DFS and developed taxonomy of issues in the design of DFS. A major step in the evolution of DFS's was the recognition that access to remote file could be made to resemble access to local files. This property, called network transparency, implies that any operation that can be performed on a local file may also be performed on a remote file. The extent to which an actual implementation meets this ideal is an important measure of quality. The Newcastle Connection and Cocanet [9] are two early examples of systems that provided network transparency. In both cases the name of the remote site was a prefix of a remote file name.

DFS provides location transparency and redundancy to improve data availability in the face of failure or heavy load by allowing shares in multiple different locations to be logically grouped under one folder, or DFS root. Many DFS performances are very low compared to the local file systems because they perform synchronous I/O operations for cache coherence and data safety [10]. File systems such as AFS [11] and NFS [12] present users with the abstraction of a single, coherent namespace shared across multiple clients. Although caching data on local clients improves performance, many file operations still use synchronous message exchanges between client and server to maintain cache consistency and protect against client or server failure.

Structuring a distributed system is a demanding task, even if the size of the system is quite limited. But the work becomes much more difficult when the scale of the system is very large. We need to consider that any viable distributed system architecture must support the notion of autonomy if it is to scale at all in the real world [13].

When we consider the location transparency it can be viewed as a binding issue. The binding of location to name is static and permanent when pathnames with embedded machine names are used. The binding is less permanent in a system like Sun NFS. It is most dynamic and flexible in Google and Hadoop. Usage experience has confirmed the benefits of a fully dynamic location mechanism in a large distributed environment.

Another major issue that attracts attention in the DFS is the failure of a machine (server or client), which cannot be distinguished from the failure of a communication link, or from slow responses due to extreme overloading. Therefore, when a site does not respond one cannot determine if the site has failed and stopped processing, or if a communication link has failed and the site is still operational. One must then assume that the inaccessible site is still capable of processing file requests. The file system protocol should handle this case in such a way that the consistency and semantic guarantees of the system will not be violated.

Based on this background issues we have developed our taxonomy that need to be considered when designing a DFS for grids, search engines etc. The following section covers the taxonomy in detail.

3. Taxonomy of Distributed File System

The motivation behind developing this taxonomy is to analyze the features that constitute the DFS and helps to incorporate a most appropriate and suitable file system that performs better, fault tolerant and secured one.

✧ **Architecture**

First Issue considered during the study was to find the types of DFS architectures that are available. Different DFS Architectures exists such as **Client-Server Architectures** (e.g. Sun Microsystem's Network File System) which provides a standardized view of its local file system. This old fashion of DFS comes with a communication protocol that allows clients to access the files stored on a server thus allowing a heterogeneous collection of processes running on different operating systems and machines share a common file system. Advantage of this scheme is that it is largely independent of local file systems. Important issue is that it cannot be used in MS-DOS due to its short file names. Another type of Architecture is **Cluster-Based Distributed File System** such as Google File Systems. It consists of a Single master along with multiple chunk servers and divided into chunks of 64 Mbytes each. The advantage

is its simplicity and it allows single master to control a few hundred chunk servers. In the Cluster-based DFS, there are three important features of architecture that usually be considered during design are *Decoupled metadata and data, Reliable Autonomic Distributed Object Storage,* and *Dynamic Distributed Metadata Management*. Third type of architecture is **Symmetric Architecture** that is based on peer-to-peer technology. It uses a DHT based system for distributing data, combined with a key based lookup mechanism. In a symmetric file system, the clients also host the metadata manager code, resulting in all nodes understanding the disk structures. In contrast, an **Asymmetric Architecture** file system is a file system in which there are one or more dedicated metadata managers that maintain the file system and its associated disk structures. Examples include Panasas ActiveScale, Lustre and traditional NFS file systems. Finally, a **Parallel Architecture** file system is one in which data blocks are striped, in parallel, across multiple storage devices on multiple storage servers. Support for parallel applications is provided allowing all nodes access to the same files at the same time, thus providing concurrent read and write capabilities. Most of the current DFS's support this important feature. An important note is that all of the above definitions overlap. A DFS can be *symmetric* or *asymmetric*. Its servers may be *clustered* or *single* servers. And it may support *parallel* applications or it may not. Based on the survey it is been identified that Multiple layers architecture allows flexibility so that protocol or functional layers can be easily added.

✧ **Processes**

Even though DFS's processes have no unusual properties the important aspect concerning this is whether they should be **stateless** or not. The primary advantage of the stateless approach is simplicity. But it will be difficult to follow during implementation because locking a file cannot be done easily by a stateless server. Processes in some of the most commonly used DFS's are studied and its flaws are analyzed. Except PVFS2, almost other DFS's support **stateful** processes. The major advantage of a *stateless* architecture is that clients can fail and resume without disturbing the system as a whole. This feature allows PVFS2 to scale to hundreds of servers and thousands of clients without being impacted by the overhead and complexity of tracking file state or locking information associated with these clients.

✧ **Communication**

Most of the DFS's use Remote Procedure Call method to communicate as they make the system independent from underlying operating systems, networks and transport protocols. In RPC approach, there are two communication protocols to consider, which are **TCP** and **UDP**. TCP is mostly used by all DFS's. However, UDP is also considered for improving performance in Hadoop. There is also a completely different approach to handle communication in DFS is Plan 9. It is mainly a file-based distributed system and in this all resources are accessed in the same way, namely with file like syntax and operations, including even resources such as processes and network interfaces. In this aspect, Lustre has considered a more flexible architecture in which they can provide **Network Independence**. Lustre can be used over a wide variety of networks due to its use of an open Network Abstraction Layer. Therefore, it provides unique support for heterogeneous networks.

✧ **Naming**

It plays an important role as each object has an associated logical path name and physical address. An aggregation of all the logical path names comprises a distributed name space which can be logically partitioned into domain. The addresses of the objects are used to access the objects in order to retrieve information from the distributed system. The *naming structure* of the file system, the *application programming interface*, the *mapping* of the file system abstraction on to physical storage media, and the *integrity* of the file system across power, hardware, media and software failures. It is been identified in systems such as Network File System. Its fundamental idea is to provide its clients complete transparent access to a remote file system. The currently common approach employs a **central metadata server** to manage file name space. Therefore decoupling metadata and data improve the file namespace throughput and relief the synchronization problem. Another approach is **metadata distributed in all nodes** resulting in all nodes understanding the disk structure. But serious implication is users do not share name spaces due to security issues. It makes file sharing harder. The different systems are studied and analyzed to define the most appropriate naming structure and method.

✧ **Synchronization**

The vital issue that is to be analyzed in the DFS is Synchronization issue. In a distributed system, the **Semantics of File Sharing** becomes a bit tricky when performance issues are at stake. When a same file is shared by two or more users, it is necessary to define the semantics of reading and writing precisely to avoid problems. Even though it looks conceptually simple, it is quite difficult to implement. There are few approaches that are available such as *UNIX semantics,*

Session semantics, Immutable semantics, and *Transactions*. Apart from semantics, we also consider to analyze the **File Locking System** in the DFS. Depending on the purpose of applications deploying on the DFS, it is developed with different locking mechanism. Major usages require *Write-once-read-many* access model. However, there are applications such as search engines require *Multiple-producer/single-consumer* access model. GFS is the infamous example for this model. To support their access model, some systems choose to *give locks on objects to clients*, and some choose to *perform all operations synchronously on the server*. Giving locks on objects to clients lead to one performance improvement by *caching at client*. Lustre is the one that apply *hybrid* solution for File Locking System. In Lustre, Locking mode is chosen differently depending on the resource contention level. The last issue we study in synchronization problem is using **leases,** which is the most common method to control the parallel access to DFS.

✧ **Consistency and Replication**

To provide the consistency, most of DFS employ **checksum** to validate the data after sending through communication network. Besides, **Caching** and **Replication** play an important role in DFS, most notable when they are designed to operate over wide-area network. It can be done in quite few ways such as **Client-side caching** and **Server-Side replication**. There are two types of data need to be considered for replication: *metadata replication* and *data object replication*. Metadata is the most important part of the whole DFS. Thus, all DFS provide a mechanism to ensure the availability and recoverability of this data such as *backup metadata server* and *snapshot of metadata with transaction logs*. For data objects, there are different approaches depending on the purpose of applications. DFSs like Lustre and Panasas assume that data object is available as long as the physical devices are available. Hence, they consider a *physical failure as exception* and the object data can be lost. However, there are some systems like Lustre which supports *RAID0 model* to store data to reduce the probability of loosing data and increase the access performance. In case of other DFSs like GFS and Hadoop, their applications require the availability of data as the critical condition and *failure will be the norm* rather than the exception. Thus, data objects are replicated in different servers. This high bandwidth consuming feature leads to the **asynchronous replication** method name "Replication in pipeline" which is employed in GFS and Hadoop.

✧ **Fault Tolerance**

Fault tolerance is very much related to the replication feature because replication is created to provide availability and support transparency of failures to users. As mentioned in Consistency and Replication section, there are two approaches for fault tolerance on object data: **failure as exception** and **failure as norm**. "Failure as exception" systems will *isolate the failure node* or *recover the system from last normal running state*. "Failure as norm" systems employ replication of all kind of data and execute re-replication whenever replication ratio becomes unsafe.

✧ **Security**

Authentication Issues and access control are some of the important security issues in DFS's that need to be analyzed. Impact of decentralized authentication is also taken into consideration during the survey of the DFS's. Most DFS employ **security with authentication, authorization and privacy** by leveraging existing security systems. Yet, some DFS's for specific purposes such as GFS and Hadoop, base on the trust between all nodes and clients so that they don't employ **no dedicated security mechanism** in their architecture.

✧ **Other Issues**

One important issue of DFS's is the ability **to use commodity devices** to build up the system. The advantage of this capability is whenever the commodity devices are improved, the DFS is automatically and naturally improved. Besides, it also become very cost effective when there is the need to scale up the system.

4. Comparison of Distributed File Systems

The systems surveyed are Google File System (GFS) [15], Lustre [16], Kosmos File System [17], Hadoop Distributed File System (Hadoop) [14], Panasas [18], Parallel Virtual File System (PVFS2) [19], and Redhat Global File System (RGFS) [20]. There are many more DFS in the literature such as NFS, AFS, QFS, and ZFS… However, due to space limitation, their novelty and their representative, we could not add them into our survey. We summarize the comparison in Table 1.

Table 1: Overall Comparison of Different Distributed File Systems

File system	GFS	KFS	Hadoop	Lustre	Panasas	PVFS2	RGFS
Architecture	Clustered-based, asymmetric, parallel, object-based	Clustered-based, asymmetric, parallel, object-based	Clustered-based, asymmetric, parallel, object-based	Clustered-based, asymmetric, parallel, object-based	Clustered-based, asymmetric, parallel, object-based	Clustered-based, symmetric, parallel, aggregation-based	Clustered-based, symmetric, parallel, block-based
Processes	Stateful	Stateful	Stateful	Stateful	Stateful	Stateless	Stateful
Communication	RPC/TCP	RPC/TCP	RPC/TCP&UDP	Network Independence	RPC/TCP	RPC/TCP	RPC/TCP
Naming	Central metadata server	Central metadata server	Central metadata server	Central metadata server	Central metadata server	Metadata distributed in all nodes	Metadata distributed in all nodes
Synchronization	Write-once-read-many, Multiple-producer/single-consumer, give locks on objects to clients, using leases	Write-once-read-many, give locks on objects to clients, using leases	Write-once-read-many, give locks on objects to clients, using leases	Hybrid locking mechanism, using leases	Give locks on objects to clients	No locking method, no leases	Give locks on objects to clients
Consistency and Replication	Server side replication, Asynchronous replication, checksum, relax consistency among replications of data objects	Server side replication, Asynchronous replication, checksum	Server side replication, Asynchronous replication, checksum	Server side replication – Only metadata replication, Client side caching, checksum	Server side replication – Only metadata replication	No replication, relaxed semantic for consistency	No replication
Fault tolerance	Failure as norm	Failure as norm	Failure as norm	Failure as exception	Failure as exception	Failure as exception	Failure as exception
Security	No dedicated security mechanism	No dedicated security mechanism	No dedicated security mechanism	Security in the form of authentication, authorization and privacy	Security in the form of authentication, authorization and privacy	Security in the form of authentication, authorization and privacy	Security in the form of authentication, authorization and privacy

5. Findings

Based on the survey and taxonomy, the following findings on different DFS's can help to select an appropriate DFS according to the application and the requirements.

✧ **Lustre:**

Lustre is a shared disk file system. Commonly used for large scale cluster computing. It is an open-standard based system with great modularity and compatibility with interconnects, networking components and storage hardware. It is suitable for general purposes file systems. Currently, it is only available for Linux.

✧ **Kosmos File System:**

Kosmos Distributed File System (KFS), a high performance DFS that supports applications whose workload could be characterized as, Primarily write-once/read-many workloads, Few millions of large files, where each file is on the order of a few tens of MB to a few tens of GB in size, Mostly sequential access. It provides high performance combined with availability and reliability. It is intended to be used as the backend storage infrastructure for data intensive apps such as, search engines, data mining, grid computing etc.

✧ **Hadoop:**

Hadoop is a Distributed parallel fault tolerant file system. It is designed to reliably store very large files across machines in a large cluster. It is inspired by the Google File System. Hadoop DFS stores each file as a sequence of blocks; all blocks in a file except the last block are the same size. Blocks belonging to a file are replicated for fault tolerance. The block size and replication factor are configurable per file. Files are "write once" and have strictly one writer at any time.

✧ **Google file system:**

Google File System is a proprietary DFS developed by Google for its own use. It is designed to provide efficient, reliable access to data using large clusters of commodity hardware. In GFS files are huge by traditional standards and are divided into chunks of 64 megabytes. Most files are mutated by appending new data rather than overwriting existing data: once written, the files are only read and often only sequentially. It is also optimized to run on computing clusters, the nodes of which consist of cheap, "commodity" computers, which means precautions must be taken against the high failure rate of individual nodes and the data loss.

✧ **Panasas**:

It implements file system entirely in hardware. It is suitable for general purposes file systems. To improve overall utilization of storage systems, network performance and increasing access to vital data, Panasas has developed ActiveScale Storage cluster. By combining a DFS with smart hardware, the Panasas Storage Cluster scales dramatically in both capacity and performance and extends appliance-like ease-of-manageability to a virtually boundless storage system.

✧ **PVFS2**:

The data access is achieved without file or metadata locking. PVFS2 is best suited for I/O-intensive (i.e., scientific) applications. PVFS2 could be used for high-performance scratch storage where data is copied and simulation results are written from thousands of cycles simultaneously.

✧ **RGFS**:

It is an open-standard based system with great modularity and compatibility with interconnects, networking components and storage hardware. Besides, it is a relatively low-cost, SAN-based technology. It is suitable for general purposes file systems. However, it is only available on Red Hat Enterprise Linux.

6. Conclusion

The DFS is one of the most important and widely-used form of shared permanent storage. The continuing interest in DFS bears testimony to the robustness of this model of data sharing. As elaborated in the preceding section, architecture, naming, synchronization, availability, heterogeneity and support for databases will be key issues that are to be taken into consideration while designing the DFS. Security will continue to be a serious concern and may, in fact, turn out to be the big concern for large distributed systems. In this paper, taxonomy was developed for the DFS and based on the taxonomy some of the most popular and common distributed file were reviewed and surveyed. The features, its advantages and disadvantages of each DFS are outlined and in detailed and also outline the findings that enables to select an appropriate one according to their needs.

7. References

[1] Chandramohan A. Thekkath, et al, "Frangipani: A scalable Distributed File System", System Research Center, Digital Equipment Corporation, Palo Alto, CA, 1997.

[2] Barbara Liskov, et al, "Replication in the Harp File System", Laboratory of Computer Science, MIT, Cambridge, CA, 1991.

[3] John Douceur and Roger Wattenhofer, "Optimizing file availability in a server-less distributed file system" In Proceedings of the 20th Symposium on Reliable Distributed Systems, 2001.

[4] Eliezer levy and Abraham silberschatz, "Distributed File Systems: Concepts and Examples", ACM Computing Surveys, Vol. 22, No. 4, December 1990..

[5]Yasushi Saito and Marc Shapiro, "Optimistic Replication", ACM Computing Surveys, Vol. 37, No. 1, March 2005, pp. 42-81.

[6] Satyanarayanan, M., "A Survey of Distributed File Systems," Technical Report CMU-CS-89- 116, Department of Computer Science, Camegie Mellon University, 1989

[7] Howard, J.H., et al, "Scale and Performance in a Distributed File System," ACM Transactions on Computer Systems, Vol. 6, Issue 1, February 1988.

[8] Nelson, M.N., et al. "Caching in the Sprite Network File System," ACM Transactions on Computer Systems, February, 1988

[9] Rowe, L.A., Birman, K.P. "A Local Network Based on the Unix Operating System", IEEE Transactions on Software Engineering SE-8(2), March, 1982.

[10] Edmund B. Nightingale, Peter M. Chen, and Jason Flinn, "Speculative Execution in a Distributed File System", ACM SOSP'05, October 23–26, 2005, Brighton, United Kingdom.

[11]Howard,J.H.,Kazar,M.L.,Menees,S.G.,Nichols, D. A., Satyanarayanan, M., Sidebotham, R. N., and West, M. J. "Scale and performance in a distributed file system", ACM Transactions on Computer Systems, Vol. 6, Issue1, February 1988

[12] Callaghan, B., Pavlowski, B., and Staubach, P., "NFS Version 3 Protocol Specification", Technical Report RFC 1813, IETF, June 1995.

[13] Alonso, Rafael and Luis L. Cova, "Resource Sharing in a Distributed Environment," Proceedings of ACM SIGOPS European workshop, Cambridge, England, September, 1988.

[14] The Hadoop Distributed File System http://hadoop.apache.org/core/docs/current/hdfs_design.html

[15] Ghemawat, S., Gobioff, H., Leung, S.T., "The Google file system", ACM SIGOPS Operating Systems Review, Volume 37 , Issue 5, pp. 29-43, December, 2003.

[16] Braam, P.J, "The Lustre storage architecture", White Paper, Cluster File Systems, Inc., October, 2003.

[17] "KOSMOS DISTRIBUTED FILE SYSTEM", http://kosmosfs.sourceforge.net/

[18] Nagle, D., Serenyi, D., Matthews, A., "The Panasas ActiveScale Storage Cluster: Delivering Scalable High Bandwidth Storage", Proceedings of the 2004 ACM/IEEE conference on Supercomputing, pp. 53-, 2004.

[19] Yu, W., Liang, Sh., Panda, D.K., "High performance support of parallel virtual file system (PVFS2) over Quadrics", Proceedings of the 19th annual international conference on Supercomputing, pp. 323-331, 2005.

[20] "Red Hat Global File System", White Paper, www.redhat.com/whitepapers/rha/gfs/GFS_INS0032US.pdf.