ISSN:0975-9646

Distributed File Systems: A Survey

L.Sudha Rani, K. Sudhakar , S.Vinay Kumar

Assistant Professor, Computer Science Department,
GPREC, Kurnool.

Abstract: **A wide variety of applications such as aerodynamic research, weather forecasting, scientific applications relies on distributed environments to process and analyse large amounts of data. As the amount of data increases, the need to provide efficient, easy to use solutions has become one of the main issues for these type of computations. The best solution to this issue is the use of Distributed File Systems (DFSs). The main aim of a distributed file system (DFS) is to allow users of physically distributed computers to share data and storage resources by using a common file system. There are many projects which focused on network computing that have designed and implemented distributed file systems with different architectures and functionalities. In this paper, we develop a comprehensive taxonomy for describing distributed file system architectures and use this taxonomy to examine existing distributed file systems implementations in very large-scale network computing systems**

Keywords: Distributed File System(DFS), AFS, NFS, GFS, HDFS.

I. INTRODUCTION

Distributed File System is a special case of distributed system. In the Distributed file system, storage resources and clients are dispersed in the network. The main goal of Distributed File System is to provide common view of centralized file system, even though it has a distributed implementation. In a distributed file system, one or more central servers store files that can be accessed by any number of remote clients in the network. The DFS allows multi-computer systems to share files without any need of IPC or RPC. Files are shared between users in a hierarchical and unified view.

Distributed File System has different requirements when compared to that of local file system. The following are the requirements which are to be considered when designing the Distributed File System.

- The fault tolerance feature must be well-implemented. How fast the data can be recovered after any failure becomes one of the most important requirements here.
- Files stored in DFS will be very huge. Most of the files' size exceed GB level. Handling these type of huge files is very crucial in DFS. Some FS will divide files into blocks. The advantage by doing this is downgrading the size of data handled by one operation from several GBs to several MBs. But on the other hand, it requires additional mapping procedure for every operation, which may cause performance drop.
- Most of the files in DFS are in write-once-read-many pattern. Therefore many DFS' provide optimized function for file writer and reader. Few of them also have efficient function to edit an arbitrary position in an existing file. Some DFS' don't even provide function to change any existing file.
- Metadata plays a key role in DFS. Since most DFS has the support for millions of files, it's not possible to efficiently retrieve the information on any given file simply by traversing every node directly. Due to this reason, most DFS assign a certain node as the central, which maintains the metadata of all files stored in the system. The retrieval for file information will become much faster via the metadata list.

The structure of the paper is as follows: Section II covers issues in designing a distributed system. In section III, Taxonomy of Distributed File System is reviewed in detail. In Section IV, overview of different Distributed File Systems is given In Section V, Conclusion of the paper is outlined.

II. ISSUES IN DISTRIBUTED FILE SYSTEM [1]

There are various points that have to be taken into consideration while designing a distributed file system.

The different issues are: Transparency, flexibility, reliability, performance, scalability, security which are described in this section.

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.The different types of transparencies are as follows:

Table 1.Types of Transparencies [1]

Transparency	Description
Access	Hides the differences in data representation and how a resource is accessed.
Location	Hides the physical location of a resource.
Migration	Hides the movement of a resource to another location.
Relocation	Hides the movement of a resource to another location while in use.
Replication	Hides the fact that multiple copies of the resource exist without user's knowledge

B. Flexibility

The best way to achieve flexibility is to take a decision whether to use monolithic kernel or microkernel on each machine. The major functions of kernel are: memory management, process management and resource management.

Monolithic kernels use the " kernel does it all" approach with all functionalities provided by the kernel irrespective of whether all machines use it or not.

On the other hand, micro-kernels use the minimalist, modular approach with accessibility to other services as needed.

C. Reliability

The users prefer a distributed system where multiple processes are available as it guards against single-processor system crashes. Thus on failure, a backup is available. Reliability means that data should be available without any errors. In case of replication, all copies should be consistent.

D. Performance

It means that an application should run just as it were running on a single processor. The metrics used for measuring performance are: response time, throughput, system utilization and amount of network capacity used.

Message transmission over a LAN takes some time, about one millisecond. To optimize performance, reduce the number of messages transmitted.

For example, a very small computation like addition of two numbers need not be computed remotely.

E. Scalability

Distributed systems are designed to work with a few hundred CPUs. There may be a need to expand the distributed system by adding more CPUs. To support more users or resources, there are limitations with centralized service, tables and algorithms.

The issues of scalability can be summarized as

Table 2.Scalability related issues [1]

Concepts	Examples
Centralized Services	A single server for all users
Centralized Data	A single online telephone book
Centralized Algorithms	Doing routing based on complete information

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.

III. TAXANOMY OF DISTRIBUTED FILE SYSTEMS [2]

Analyzing the features that constitute the DFS helps to incorporate most appropriate and suitable file system. Taxonomy is done based on various factors which are listed below:

A. Architecture

The following are different DFS architectures that exists

1. **Client- Server Architecture**: Sun Microsystems's Network System which provides standardized view of local file system.
2. **Cluster-Based Distributed** File System such as GFS. It consists of a single master along with multiple chunk servers and divided into multiple chunks.
3. **Symmetric Architecture:** In this file system, the clients also host the metadata manager code, resulting in all nodes understanding the disk structures. This is based on peer-to-peer technology.
4. **Asymmetric Architecture:** There are one or more dedicated metadata managers that maintain the file system and its' associated disk structures. Examples include Luster and traditional NFS file systems.
5. **Parallel Architecture:** Here, data blocks are placed in parallel, across multiple storage devices on multiple storage servers. Support for concurrent read and write capabilities.

B .Processes

The important aspect concerning is whether processes should be **stateless** or not. The primary advantage of the stateless approach is simplicity.

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.

C. Communication

DFS's use Remote Procedure Call (RPC) method to communicate as they make the system independent from underlying OS, networks and transport protocols. In RPC approach, there are two communication protocols to consider, TCP and UDP.

1. **TCP** is mostly used by all DFS's.
2. **UDP** is considered for improving performance in Hadoop.

There is also a completely different approach to handle communication in DFS which is a file based distributed system. In this, all the resources are accessed in the same way with file like syntax.

Luster provides **Network Independence**, thus it can be used on a wide variety of networks due to its use of an open Abstraction Layer.

D. Naming

The currently common approach employs

1. **Central metadata server** to manage file name space. Therefore decoupling metadata and data improve the file name space and relief the synchronization problem.
2. **Metadata distributed in all nodes** resulting in all nodes understanding the disk structure.

E. Synchronization

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. Apart from semantics, we also consider to analyze the **File Locking System** in the DFS.

Major usages require

Write-once-read many access models. However, there are applications such as search engines require **multiple producer/single-consumer** access models.
Some systems choose to give locks on objects to clients and some choose to perform all operations synchronously on the server.
Lustre applies *hybrid* solution for File Locking System.Using **leases** is the most common method to control the parallel access to DFS.

F. Caching and Replication
Most of DFS employ checksum to validate the data after sending through communication network. Caching and Replication play an important role in DFS when they are designed to operate over wide area network. It can be done in many ways such as

1. Client-side caching
The client asks the server if the cached data is ok

2. Server-Side replication.
The server is notified on every open. If a file is opened for writing, then disable caching by other clients for that file. There are two types of data that need to be considered for replication: metadata replication and data object replication.

G. Fault Tolerance
It is very much related to the replication feature because replication is created to provide availability and support transparency of failures to users. There are two approaches for fault tolerance: failure as exception and failure as norm.

1. **Failure as exception systems** will isolate the failure node or recover the system from last normal running state.
2. **Failure as norm systems** employs replication of all kinds of data.

IV. OVERVIEW OF DISTRIBUTED FILE SYSTEMS [3]

NETWORK FILE SYSTEM (NFS) [4]

History
Network File System(NFS) is developed by Sun Microsystems in 1985.It is the most popular, open, and widely used distributed file system for many years. NFS has 2 versions NFSv3: version three-used for many years and NFSv4: introduced in 2003 Version 4 made significant changes to changes to NFS3.
NFS implements a file system model that is almost identical to a UNIX system. Files are structured as a sequence of bytes and the file system is hierarchically structured. It supports hard links and symbolic links. It implements most of the file operations which are supported by

A. NFS goals:
- **E**ach file server presents a standard view of its local file system.
- Transparent access to remote files.
- It has to be compatible with multiple platforms and operating systems.
- Easy crash recovery at server.

B. Architecture:
NFS has Client-Server Architecture. It provides a standardized view of the local file system. Clients access the server transparently through an interface similar to the local file system interface. Client-side caching may be used to save time and network traffic. Server defines and performs all file operations.
Virtual File System (VFS) acts as an interface between the operating system's system call layer and all file systems on a node. The user interface to NFS is the same as the interface to local file systems. The call actually goes to the VFS layer, which forwards that either to the local file system or to the NFS client.

C. Processes:
NFS servers historically did not retain any information about past requests. The consequence is that crashes weren't too painful. If server is crashed, it had no tables to rebuild so the server can just reboot and start again. The disadvantage is that client has to maintain all state information. Therefore messages are longer than they would be otherwise. NFSv4 is state*ful.*

D. Communication:
The NFS client communicates with the server using RPCs File system operations are implemented as remote procedure calls. At the server: an RPC server stub receives the request, "un-marshalls" the parameters & passes them to the NFS server, which creates a request to the server's VFS layer. The VFS layer performs the operation on the local file system and the results are passed back to the client.

E. Synchronization:
Synchronization of file contents (one-copy semantics) is not guaranteed when two or more clients are accessing the same file.
NFSv3 offers the following two strategies for updating the disk:
Write-through – All the altered pages are written to disk as soon as they are received to the server. When a write() call is made, the NFS client comes to know that the page is on the disk.
Delayed commit – The pages are placed in the cache until the commit() call is received for the relevant file. This is the default mode used by NFSv3 clients. A commit() is issued by the client whenever the file is closed.

F. Caching and Replication:
Server Caching
This is similar to UNIX file caching for local files:
pages (blocks) from disk are held in a main memory buffer cache until the space is required for newer pages. Read-ahead and delayed-write optimizations.
For local files, writes are deferred to next sync event (30 second intervals).Server caching works well in the local context.
Client Caching
The Server caching does not reduce the RPC traffic between client and server. Further optimization is needed to reduce the server load in the case of large networks.

NFS client module caches all the results of read, write, getattr, lookup and readdir operations.

G. Fault Tolerance:
Fault tolerance provided in NFS is limited but it is effective. Services are suspended whenever the server fails. Recovery from the failures is aided through the stateless design.

ANDREW FILE SYSTEM (AFS)

History
Andrew File System (AFS) is a file system that was developed as a part of a larger project known as Andrew. AFS was originally developed for computers which run operating systems such as BSD UNIX and Mach. Like NFS, AFS provides transparent access to remote shared files for UNIX programs running on workstations.AFS is implemented as two software components that exist at UNIX processes called Vice and Venus. The research on AFS is now done in a project named as Open AFS. The Open AFS implementation supports a number of platforms such as: Linux, Apple Mac OSX, Sun Solaris and Microsoft Windows NT.

A. Goals
- The main design goal of AFS was to create a system for large networks.
- Transparent access to remote shared files.

B. Architecture
The basic organizational unit of AFS is the *cell*. An AFS cell is the collection of server and client machines which is administered independently.. Since AFS is the most commonly used DFS in various academic and research environments, the cells should also participate in a global AFS file tree. However, this is not mandatory.
Servers and clients can only belong to a single cell at any given time, but the user may have accounts in multiple cells. The cell that is first logged in to is known as the *home cell* and the other cells are known as the *foreign cells*. From the user's point of view, the AFS hierarchy is accessed through the path /afs/ [cell name] by using normal file system operations. Since AFS is location transparent, the path to a file in the tree will be the same regardless of the location of the client. Whether the file is actually accessible or not depends on the user's credentials and the file's access permissions, as set by Access Control Lists.

C. Processes:
In AFS neither the server nor the client is stateful. They don't store any sort of information regarding the requests made by the clients..

D. Communication:
AFS communication is implemented over TCP/IP. The RPC protocol named Rx, which is developed for AFS, is used for communication between two machines.The communication protocol is optimized for wide area networks, and there are no restrictions on the geographical locations of participating servers and clients.

E. Synchronization:
Generally AFS is dependent on server clock synchronization for database replication. This is achieved by using the Network Time Protocol Daemon. One server acts as a synchronization site, getting the time from an external source. The other servers update their clocks against the synchronization site.

F. Caching and Replication:
The strategy that is chosen for caching in AFS was to **cache files** locally on the **clients**. When a file is opened, the Cache Manager (client-side component) initially checks whether there is a valid copy of the file in the cache or not. If the file is not there in the cache, then the client retrieves the file form the server by making a request to it. A file that has been modified and closed on the client is transferred back to the server. The client builds up a "working set" of often-accessed files in the cache. The optimal cache size is usually 100 MB, but this depends on what the client is used for. As long as the cached files are not modified by other users, they do not have to be fetched from the server when subsequently accessed. This reduces the load on the network significantly. Caching can be done on disk or in memory.

GOOGLE FILE SYSTEM (GFS) [5]

History
The Google File System (GFS) is introduced in 2003 to meet the rapidly growing demands of Google's data processing requirements. It is a scalable distributed file system which was especially developed for large distributed data-intensive applications. It provides fault tolerance while running on inexpensive commodity hardware, and it delivers high aggregate performance to a large number of clients. GFS clusters consists of hundreds or even thousands of storage machines that are built from inexpensive commodity hardware.

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. Architecture
GFS employs cluster based architecture. A GFS cluster consists of a single *master node* , multiple *chunk servers* and is accessed by multiple *clients*. Files are divided into fixed-size *chunks*. Each chunk is identified by using an unique 64 bit *chunk handle* assigned by the master at the time of chunk creation. For reliability, each chunk is replicated on multiple chunk servers. By default, the replication factor is chosen as three.
The master node maintains all the metadata. This includes the access control information, the mapping from files to

chunks, and the current locations of chunks. It also controls system-wide activities such as chunk lease management, garbage collection of orphaned chunks, and chunk migration between chunk servers. The master periodically communicates with each chunk server through *Heart Beat* messages to collect their state.

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

D. Communication
Clients send data request to the master node, which maintains metadata of all chunks and the mapping from file to chunks. The master returns the metadata requested to the client. Then the client is enabled to connect to the chunk server directly for data transfer. All this communication in between the master node and client nodes is performed using RPC/TCP
Protocol.

E. Synchronization
GFS is equipped with a carefully designed locking function that can handle multi-operation to one same chunk simultaneously.

F. Caching and Replication
The GFS uses chunk replica placement policy .It serves two purposes: maximize data reliability and availability, and maximize network bandwidth utilization. For both of these purposes it is not enough to spread replicas across machines, which can guard only against disk or machine failures and fully utilizes each machine's network bandwidth. We must also spread chunk replicas across racks. This ensures that some replicas of a chunk will survive and remain available even if an entire rack is damaged or offline.

G. Fault Tolerance
All the data in GFS is triple replicated. Whenever a chunk server is down, the master can always redirect data requests to the replicas, until the node is back online. If the master fails, the system can easily choose another node to generate a metadata list by scanning over all chunk servers and work as master.

HADOOP DISTRIBUTED FILE SYSTEM (HDFS) [6][7]

History
The Hadoop Distributed File System (HDFS) is an open-source version of GFS from Yahoo. The HDFS is a distributed file system which is designed to run over commodity hardware. It has many similarities with the existing distributed file systems. The significant difference of HDFS with other distributed systems is that, HDFS is highly fault-tolerant. HDFS provides high throughput access to application data and is especially designed for applications that have large data sets.

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.
- HDFS has to provide high aggregate data bandwidth and it has to scale to hundreds of nodes in a single cluster.
- It should support tens of millions of files in a single instance.
- Detection of faults , quick and automatic recovery from them is a core architectural goal of HDFS.

B. Architecture
HDFS employs a master/slave architecture. An HDFS cluster consists of a single Name Node, which is the master node that manages the file system namespace and regulates the access to files by clients. In addition to the Name Node there are number of Data Nodes, which manage the storage attached to the nodes that they run on. The user data is stored in the form of files, generally a file is split into one or more blocks and these blocks are stored in a set of Data Nodes.
The Name Node executes various file system operations like opening, closing, and renaming files and directories. The Data Nodes are responsible for serving read and write requests from the file system's clients. The Data Nodes also performs block creation, deletion, and replication whenever the Name Node instructs.
The Name Node and Data Node are pieces of software that are designed to run on commodity machines. These machines typically run a GNU/Linux operating system (OS). HDFS is built using the Java language, any machine that supports Java can run the Name Node or the Data Node software.

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

D. Communication
All HDFS communication protocols are layered on top of the TCP/IP protocol. A client establishes a connection the Name Node through configurable TCP port . The Data Nodes talk to the Name Node using the Data Node Protocol. A Remote Procedure Call (RPC) abstraction wraps both the Client Protocol and the Data Node Protocol. The Name Node never initiates any RPCs, it only responds to RPC requests issued by Data Nodes or clients.

E. Synchronization
HDFS applications use a write-once-read-many access model for files. Files in HDFS are write-once and have strictly one writer at any time. A file once created, written, and closed need not be changed. A Map/Reduce application or a web crawler application fits perfectly with this model. There is a plan to support for appending-writes to files.

F. Caching and Replication
HDFS is designed to store very large files across machines in a large cluster. Files are divided into blocks and each block of a file is replicated for fault tolerance. The block size and replication factor are configurable per file. The

replication factor can be specified at file creation time and can be changed later. The Name Node makes all decisions regarding replication of blocks. It periodically receives a Heartbeat and a Block report from each of the Data Nodes in the cluster. Receipt of a Heartbeat implies that the Data Node is functioning properly.

HDFS uses rack-aware replica placement policy to improve data reliability, availability, and network bandwidth utilization. HDFS's placement policy is to put one replica on one node in the local rack, another on a different node in the local rack, and the last on a different node in a different rack. This policy cuts the inter-rack write traffic generally improves write performance. The chance of rack failure is far less than that of node failure.

Communication between two nodes in different racks has to go through switches. In most cases, network bandwidth between machines in the same rack is greater than network bandwidth between machines in different racks. A simple but non-optimal policy is to place replicas on unique racks.

G. Fault Tolerance

As the HDFS's placement policy puts one replica on one node in the local rack, another on a different node in the local rack, and the last on a different node in a different rack. If the first node in the local rack fails request is sent to a different node in the same rack where the replica is found. If that node also fails then the request is sent to a different node in the different rack. Therefore due to the usage of replica placement policy, fault tolerance is achieved in HDFS.

V. CONCLUSION

The DFSs are the most important and widely used forms of shared permanent storage. DSFs are the principle storage solution used by supercomputers, clusters and data centers. Architecture, naming, synchronization, availability, heterogeneity and support for databases will be key issues that are to be taken into consideration while designing the DFS. In this paper, taxonomy was developed for the DFS and based on that taxonomy. Some of the most popular and common distributed file systems like AFS, NFS, GFS and HDFS were reviewed and surveyed.

NFS was the most popular, open, and widely used Distributed File System for many days. It is compatible with multiple operating systems and platforms. As NFS Server is stateless easy crash recovery is the most important advantage of NFS.

AFS employs aggressive client side caching with proactive cache-invalidation. This is one of the strengths of AFS. But AFS has some Weaknesses like it is difficult to setup, very complex, it has a big Linux kernel module for client. Due to these reasons AFS could not become as much popular as NFS.

For data intensive applications where massive amount of data has to be stored DFSs like GFS and HDFS can be used.

GFS is fully distributed. This is the important feature of DFS. But it has some weaknesses like it requires heavy-duty, non-standardized cluster management system. and it is compatible only with Linux platform. The negative side of GFS is that it is not a open source version. It is especially designed for Google Applications and can be used only by the people of Google.

HDFS is the open source version and can be used by everyone. It offers tremendous opportunity for massive scale. It can be very useful for commercial applications like Marketing analytics, processing of XML messages etc. It is a fault tolerant distributed system and also highly scalable. Hadoop can also be integrated with resource management cloud software. As HDFS has the above mentioned advantages, current research work has been going on it. The negative point of Hadoop is that most of the organizations are trying to pull it different directions.

REFERENCES

[1] Sunita Mahajan"*Distributed Computing*", Oxford University Press
[2] Andrew Tenanbum,"*Distributed System*", PHI
[3] Tran Doan Thanh, "*A Taxonomy and Survey on Distributed File Systems* ", Fourth International Conference on Networked Computing and Advanced Information Management.
[4] Russell Sandbaerg, "*Design and Implementation or the SUD Network File system*", Sun Microsystems.
[5] 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.
[6] Konstantin Shvachko," *The Hadoop Distributed File System*", Yahoo-Inc.com.
[7] The Hadoop Distributed File System http://hadoop.apache.org/core/docs/current/hdfs_design.html