Evolution and Analysis of Distributed File Systems in Cloud Storage: Analytical Survey

Dharavath Ramesh*,[1], *Member IEEE*, Neeraj Patidar[2], Gaurav Kumar[3], Teja Vunnam[4]
Department of Computer Science and Engineering
Indian School of Mines, Dhanbad, India
ramesh.d.in@ieee.org*, neerajism@cse.ism.ac.in, gaurav315kumar@gmail.com, vunnamteja@gmail.com

***Abstract*—Handling of Big Data and cloud computing are the two important prime concerns which have become more and more popular in recent years. Due to tremendous hike in data production, the need for the efficient processing, transaction storage or retrieval, and management of the structured and unstructured data have become one of the important issues of the IT industry. In this paper, we examine the concept of evolution of various distributed file systems, advantages, and limitations with respect to the cloud computing paradigm. Distributed File System is a client-based application which permits its users to access, process and modify the data which is stored on a remote server as if it existed on their local systems. The Google file system (GFS), a proprietary distributed file system was developed by Google to meet its own growing big data need, which involved the utilization of commodity hardware in managing the massive generation of raw data. Inspired by GFS, Hadoop Distributed File System (HDFS) was developed, which is an open-source community project. It is designed to store large chunks of data sets consistently as well as providing high bandwidth sets for streaming data on client based applications. In this paper, we describe the GFS and HDFS models in detail and relate these two distributed file systems on the basis of their properties and characteristic behavior in different environments. We also discuss several techniques and modifications to prevent and mitigate the limitations of these file systems.**

Keywords— Cloud Computing; GFS (Google File System); HDFS (Hadoop Distributed File System); HBase

I. INTRODUCTION

Efficient data storage has been considered as a very important and indispensable part of research in the field of cloud computing and Big Data [1],[2]. The tremendous hike in the rate of data production, number of clients, and the demand of cloud computing services have made it impossible to provide the same efficiency and scalability while using the traditional storage to access and modify Big Data [3]. Some of the major challenges include scalability, robustness, efficiency, data transfer bottlenecks etc. [4] Hence, we must improve our technology and design to meet the increasing demands for high scalability and efficiency. For example, a possible solution is to improve our methods such that it can lead us to store fewer data as well as Metadata [16] required for storage, extraction, security and maintenance of Big Data [1]. At the root level, we can classify all data storing techniques and methodologies as a File System. A File system defines how files are named and organized logically for storage, retrieval and modification [3]. The file system uses this metadata [16] to store and retrieve files. Some examples of sources of metadata tags include: (i) Date and time of creation or modification of the file and (ii) Size and other file related attributes etc.

The most popular file system which is used for the cloud computing is the Distributed File System (DFS). DFS is more popular due to its flexibility and scalability. The cloud computing services relies entirely on the type and the flexibility of file system of its underlying architecture. Many popular file systems have been designed for field-specific applications like business analytics, research and experimental data storage etc.[5] The most popularly used file system for cloud computing purposes is the Distributed file system. DFS is more prevalent due to its flexibility and scalability. The cloud computing services rely entirely on the type and the flexibility of file system of its underlying architecture [5].

II. DISTRIBUTED FILE SYSTEM

Distributed File System (DFS) is a client-server based application [6] which permits its users to access, process or modify the data which is stored on a remote server as if it existed on their own systems. When a client accesses a file, the server or the host provides the user a duplicate copy of the requested file. The file itself is cached on the user's system while the data is processed and returned back to the server [7]. In DFS, multiple central servers store files, which can be accessed by several isolated users at a time in the network. With the proper authorization rights, clients only can access these files. The client is able to work with the file in the same way as if it is stored locally on their local system. If the client is finished working with the file, it is returned over the network back to the server, which stores the original or altered file for retrieval at a later time by the same or another user [7]. DFS also uses a different naming scheme to map and to keep the track of files located in different non-central servers. It arranges its files according to the hierarchical file management system.

DFS distributes the related documents over different clients by arranging a centralized storage. NFS from Sun Microsystems and DFS from Microsoft are some popular examples of distributed file systems [6]. The reason behind the extensive utilization of DFS depends on the ease of their information sharing with multiple clients concurrently by exporting file systems efficiently. The main features such a

ISBN: 978-1-5090-1666-2/16/$31.00 ©2016 IEEE

system must accomplish are: Data consistency, Uniform access, Reliability, Efficiency, Performance, Manageability, Availability, and Security [7].

A. Network file System - NFS

In 1984, the Network File System (NFS), the most pervasive distributed file system was developed by Sun Microsystems [8]. It is one of the earliest DFS which is also used very frequently. The idea behind NFS is simple, but it has a considerable number of limitations due to its simplicity. NFS was designed to provide direct access to a single logical storage volume stored on a remote machine. The NFS-based server makes a share of its local file system accessible to the external clients. Then, the clients can directly access it as though it was a part of the local hard disk of their own system. One of the most important advantages of this model is its transparency. Hence, it is not necessary for the user to be aware of the location and the other details related to the storage of files which are stored remotely on some other system. Limitations of network file system are as follows:

- **Lack of Reliability:** All the files in NFS reside on an individual machine which means that if the machine goes down (e.g. Due to power failure, configuration issues, hardware problem etc.) then it will lead to a permanent or temporary termination of all services provided by the machine to the users [8]. These situations can lead to loss of data stored on the machine.
- **Bottleneck:** In NFS, the server is overloaded easily enough if a large number of clients simultaneously try to access their files from their systems to the remotely located server. The limited processing power of the machine results in poor performance of the server in fulfilling the requests made by any of the clients.

Hence, NFS works sufficiently on a small scale [8] where the number of users is less in the network but we cannot rely on NFS to meet the increasing demands at the large scale to handle huge chunks of Big Data as well as the number of users accessing that data through the cloud-based storage as a service.

B. Andrew file System - AFS

In 1980's, Andrew File System (AFS) was introduced by the researchers of Carnegie Mellon University [9] to mitigate the problem of scalability among distributed file systems. In the initial version of AFS, the whole requested file was cached on the local disk of the clients in order to increase the performance over multiple requests for the same file on the server. Hence, the number of requests fulfilled locally reduces the overall load on the server containing that file. Finally, if the file was modified on the local system, the latest version was sent back to the server upon closing of the session. However, the first access of the un-cached file is not efficient. If the file is not modified the locally cached file which is used to improve the performance. Limitations of the AFS initial version are as follows:

- **High Path traversal time:** In order to access the file at the client side the server has to traverse the complete path from the home directory to the location of the file. This traversal takes a considerable part of the server's processing time on the other hand which can be used by the server to process the other client requests.
- **High Traffic:** In AFS, a huge amount of traffic is generated by client side in the form of validation messages to check whether the file has been modified or not [9]. This traffic reduces the efficiency of file transfers within the network.

To improve the initial version of AFS [9], two modifications have been introduced to improve the scalability and performance of the AFS;

1. **Use of File Identifier (FID):** To save the CPU time of the server in traversing the path to the location of the file, File Identifier (FID) was introduced. A FID clearly mentioned which file the server was interested in and thus reduced the load on the server.
2. **Use of Callback function:** Callback is a simple function performed by the server which informs the modification of a file to all the clients which have cached the unmodified version of the same file. Thus, the regular validation messages to check the state of the file are no more required. The comparative analysis has been depicted in **Table I**.

TABLE I. Comparative analysis of NFS and AFS

Evaluation criteria	NFS	AFS
Namespace	No shared namespace, individual namespace for all clients.	Shared global namespace
File Caching	No provision for local disk caching	Entire files are cached on the user's local disk
Scalability	Only for small scale (10-20 users)	Highly scalable as compared to NFS
Security	User ID's are used to determine the file access permission	Kerberos security is used for verification
Backup	UNIX based backup system	It has its own developed system for backup
Implementation	Solaris, AIX, FreeBSD etc.	Transarc(IBM), OpenAFS etc.

C. Google File system - GFS

In 1980's, Though the next version of AFS was a big improvement, Google was still facing the challenge of storage and processing of Big Data. In 2003, they came up with Google File System (GFS) [10] as their own solution to the problem of big data management. It is a scalable distributed file system designed to handle large distributed data-intensive applications. GFS was designed with many goals common to those of many distributed file systems. Different factors like hardware failures, throughput constraints, and latency kind instances motivate the scientists to design DFS. In designing the Google file system there are some assumptions [10] which have to be considered:

- **Economical hardware building blocks:** GFS is built on low commodity hardware components which are highly prone to failure. Hence, it should have a provision to monitor itself on a regular basis to detect and recover from its component failures.
- The system stores and processes a considerable number of large files. At the same time, Small files should also be supported for processing. But still the priority is crunching huge chunks of data i.e. high throughput.
- There are two kinds of reads which are considered in a workload:

(i) Large Streaming reads: Individual operations read nearly in **MB's** or more [11]**.** It assumes that the successive requests from the same client often read from a contiguous region of the file.

(ii) Small Random reads: It includes a random query which reads only up to a few **KB's**.

- Workloads mostly consist of large sequential writes [11] which append data to files. The files are automatically updated back in their data node for future reference. At the same time, small writes at random positions are supported but not efficient.
- **Well-defined semantics:** The semantics of the systems must be efficiently implemented and well defined to make the concurrent appending on the same file by multiple users possible.
- **High sustained bandwidth** is considered more desirable than the Low latency data access. GFS has been built by giving more preference to processing of chunks of data at a high rate.

As shown in **Fig. 1,** a GFS cluster consists of a single *Master Node* and multiple *Chunk Servers* which is accessible to *multiple clients* at a time. It runs on a Linux machine as a server process. It is easy to run both the Chunk Servers and the client on the same machine until the machine resources allow and the declination in reliability is acceptable which is generated due to the execution of flaky application code. Files are further divided into a number of fixed-size *chunks* where, each chunk is managed by the Master Node. All chunks are stored at local disks as files to read and write the data.

For reliability, each chunk is duplicated on multiple Chunk servers. The master maintains all file system metadata. This includes the namespace and access control information by mapping the chunks with their locations. GFS manages management issues like collection of garbage data from individual chunks and migration between operational chunks. The master periodically communicates with each Chunk Servers in *Heartbeat messages* [10] to give it instructions and collects the state of the chunk servers. To perform metadata related operations, clients will interact with Master Nodes. None of the clients or the Chunk servers caches file data.

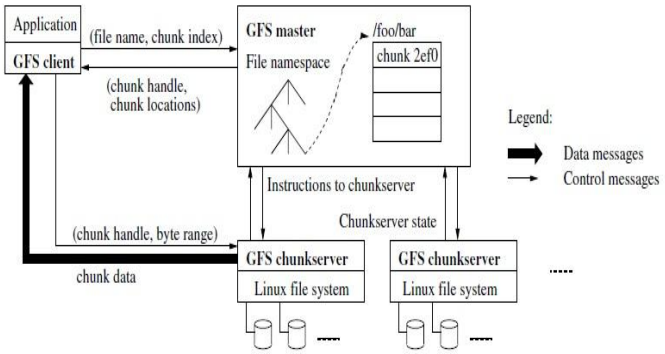

Fig. 1. Architecture of GFS ***(Source: The Google File system, [Ghemawat, Gobioff and Leung 2003][11])***

D. Hadoop Distributed File System -HDFS

Inspired from GFS, Hadoop Distributed File System (HDFS) was developed using the idea behind distributed file system design of GFS which runs on cheap commodity hardware [15]. Unlike other distributed file systems, HDFS [12] is considered as a highly robust distributed file system. It was designed using low-cost hardware which can hold a very large amount of data and provides much easier access to it [17]. In order to store such huge chunks of data, files are stored across multiple machines. These files are stored in a redundant manner to save the system from possible data losses in case of failure of the machine. It also makes the applications accessible for the purpose of parallel processing [15].

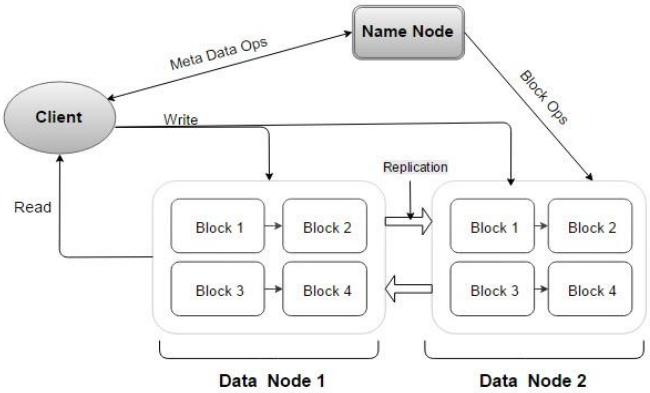

Fig. 2. Architecture of HDFS

HDFS has a master and slave architecture similar to that of NameNode and DataNode of GFS architecture [12]. A cluster is comprised of a single **Name Node** (Shown in **Fig. 2**) the main server which manages the file system's namespace and then regulates the accesses which are made to the files from clients [13]. Next, we have a number of **Data Nodes**, normally one per node in the HDFS cluster.

Hence, HDFS reveals a highly scalable and fault tolerant file system which allows the user's data to be stored efficiently in files. Internally, a file is further divided into multiple blocks. These blocks are then stored on a set of DataNodes to increase the redundancy of the data [14]. The Name Node is responsible for execution of file system namespace operations such as opening, renaming and closing of files or directories. The mapping of blocks to Data Nodes is also maintained by it. The read/write requests of the clients at remote places is fulfilled by the DataNodes. DataNodes also help in executing instructions like creation, deletion, or replication of block as per the instructions from the NameNode to the DataNodes. The comparative analysis of GFS and HDFS has been described in **Table 2**.

TABLE II. Comparative analysis of GFS and HDFS

Evaluation criteria	**GFS**	**HDFS**
Hardware	Comprises of Low commodity Linux systems with high fault chances	Low commodity hardware (preferably with high storage capacity [20])
Communication	(i)TCP connections are used (ii) Periodic commanding and synchronizing messages (heartbeats) are used	(i) RPC based protocol on top of TCP/IP is used (ii) Concept of Periodic commanding & synchronizing messages is used
Hierarchy	MasterNode and ChunkServers or SlaveNodes	NameNode and DataNode
Security	Google has lot of Data Centers at unknown location to increase the redundancy	The security of HDFS depends on POSIX model [18] of user and group
Chunk/Block Size(Default)	64 MB per chunk	128 MB per block
Content Per Block/Chunk (Metadata)	Each Chunk has 64KB data and 32-bit checksum pieces	Per HDFS block two files are created on data node, 1. *Data file* 2.*Metadata file* (checksum, timestamps)
Replication factor	3 times (Default) but it can be varied	2 or 3 times
Actively used by	Google Inc.	Yahoo!, Facebook, IBM etc.
Operations	(i) Random file writes and reads possible (ii)Multiple readers, Multiple writer model	(i) Only append is possible in HDFS (ii)Multiple readers, Single writer

III. LIMITATIONS OF GFS AND HDFS

Google file system is premeditated to store a humongous chunk of data in a reliable way so that the same can be processed commendably and resourcefully. Though it supports the facility of global namespace, efficient processing, snapshot and atomic record append, there are some limitations in the design on GFS such as:

- **Random write on a small level:** the write workload of GFS is always the big large stream. By moving some functionality out of GFS, we can get more benefit for Random writes on small files [11].
- **Fixed Chunk Size:** Sometimes the fixed chunk size of 64MB creates a problem [17]. For example, when the last chunk is divided into some parts which result in searching the whole new location to get access to the other distant parts of the file system. It also increases the communication overhead [11].

Both GFS and HDFS are not considered suitable for,

1. **Low latency data access**: As the data is stored in large chunks (64 MB) of large size, hence both GFS and its open source variant HDFS lag [20] in providing faster data access. Both are designed to crunch the large chunks of data for higher throughput.
2. **Handling files with small size:** Both GFS and HDFS are not considered suitable for handling files with small size (Less than 1MB) [19]. Even if they support operations on smaller files but it will result in a decline in efficiency.
3. **A Situation where data keeps changing frequently:** HDFS is famous for its single write multiple read scheme but still it faces problem in the situations where data is modified and updated frequently.

IV. MODIFICATIONS TO IMPROVE GFS/HDFS

A. *Using HBase to solve the problem of low latency data access*

HBase is [21], distributed, sorted map modeled after Google's big table for the faster query search in huge amount of structured data. It is open-source and scalable (Horizontally) Hadoop based database. HBase runs on top of HDFS and it is also written in java (shown in **Fig. 3**). The HDFS files are indexed in HBase for quick query response. The main purpose for HDFS is batch processing of files with large size. The low-latency data reads are not a priority of HDFS. By using HBase on top of HDFS, we can have low latency access to small files or tables from a large chunk of data.

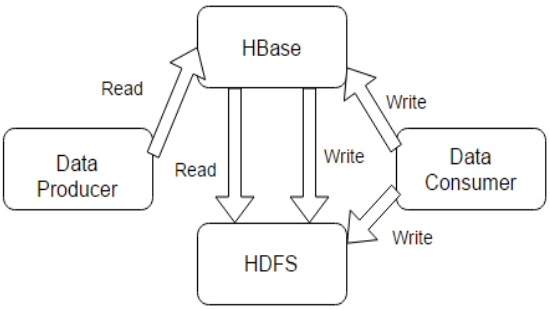

Fig. 3. HBase with Hadoop as storage [21]

B. *Improvement in Query response time using Caching*

Centralized cache management [22] is a mechanism which significantly improves the Query response time of the file system. It allows users to specify *paths* to be cached in main memory by HDFS (shown in **Fig. 4**). In this, the NameNode will communicate with its DataNodes that have the requested files on their disks, and instruct them to cache the blocks in off-heap caches [23] to improve the efficiency by directly providing the cached path for the same request by the other remote user.

Centralized cache management in HDFS has many significant advantages.

1. **Retention of frequently used data:** Caching prevents the removal of extensively used data from the memory [24]. It turns out to be useful when the size of the working set exceeds the available main memory size.
2. **Efficient read operation:** It improves the read performance by co-locating a task with a cached block replica managed by NameNode.
3. **Usage of new API:** The use of new efficient API becomes easier as the checksum verification of cached data is done earlier by the DataNode. Hence, clients will experience essentially zero overhead when using this new API.
4. **Efficient Memory utilization and management:** It improves the overall cluster memory utilization. With the facility of centralized management, a user can explicitly pin only *m* out of the *n* replicas, saving *n-m* memory where as depending on the OS buffer cache at each DataNode [22], [24].

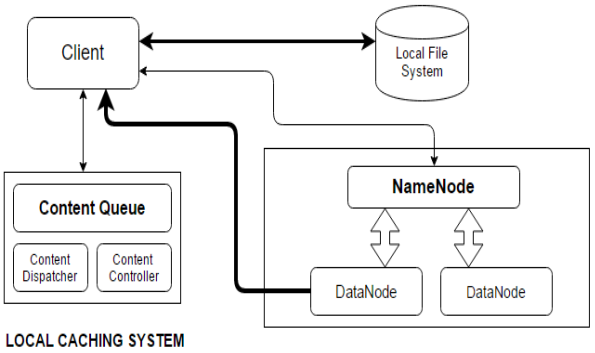

Fig. 4. HDFS with improved Caching to reduce query response time

V. CONCLUSION AND FUTURE WORK

In this paper, we have analyzed the evolution of distributed file systems, their origin, specialty and their development. We started our discussion with the analysis of Network File system and Andrew File System. We have also discussed the detailed architectures of GFS and HDFS, the two most significant distributed file systems of their time capable enough to handle the Big Data Management. A comparative study between both GFS and HDFS has been done which has resulted in the occurrence of some similarities as well as differences between both GFS and HDFS. From this analysis, we can conclude that GFS and HDFS are similar in many aspects and their implementation can be further improved. Finally, we conclude with the discussion of the limitations of GFS and HDFS and their possible effective solutions by using techniques like Centralized cache management and HBase.

Future work should look at improving the caching techniques for the cache management in the file system and to reduce the amount of MetaData to store the data without affecting its throughput and low latency data access.

ACKNOWLEDGMENT

The authors would like to express their gratitude and heartiest thanks to the Department of Computer Science & Engineering, Indian School of Mines (ISM), Dhanbad, India for providing their support.

REFERENCES

[1] Liu, K., & Dong, L. J. (2012). Research on cloud data storage technology and its architecture implementation. *Procedia Engineering*, *29*, 133-137.

[2] Marinescu, D. C. (2013). *Cloud computing: Theory and practice*. Newnes.

[3] Scott, M., Boardman, R. P., Reed, P. A., Austin, T., Johnston, S. J., Takeda, K., & Cox, S. J. (2014). A framework for user driven data management. *Information Systems*, *42*, 36-58.

[4] Hashem, I. A. T., Yaqoob, I., Anuar, N. B., Mokhtar, S., Gani, A., & Khan, S. U. (2015). The rise of "big data" on cloud computing: review and open research issues. *Information Systems*, *47*, 98-115.

[5] Barry Sosinsky, Chapter 15: Working with Cloud-Based Storage ,"Cloud Computing Bible".

[6] Distributed file system, *https://en.wikipedia.org/wiki/Distributed_File_System_(Microsoft)*

[7] Fesehaye, Debessay, Rahul Malik, and Klara Nahrstedt. "A Scalable Distributed File System for Cloud Computing." (2010).

[8] Network File System, *https://en.wikipedia.org/wiki/Network_File_System.*

[9] Andrew File System, *https://en.wikipedia.org/wiki/Andrew_File_System*

[10] Ghemawat, S., Gobioff, H., & Leung, S. T. (2003, October). The Google file system. In *ACM SIGOPS operating systems review* (Vol. 37, No. 5, pp. 29-43). ACM.

[11] UzZaman, N. (2007). *Survey on Google file system*. *Survey Paper for CSC*,*456*.

[12] Shvachko, K., Kuang, H., Radia, S., & Chansler, R. (2010, May). The Hadoop Distributed File System. In *Mass Storage Systems and Technologies (MSST), 2010 IEEE 26th Symposium on* (pp. 1-10). IEEE.

[13] Introduction to HDFS, https://www.ibm.com/developerworks/library/waintrohdfs/

[14] Hadoop Distributed File System (HDFS) Architecture guide, *https://hadoop.apache.org/docs/r1.2.1/hdfs_design.html*

[15] Tom White, Chapter 3: The Hadoop distributed file system, "Hadoop, The Definitive Guide" 4th edition, O'reilly.

[16] Romero, O., Herrero, V., Abelló, A., & Ferrarons, J. (2014). Tuning small analytics on Big Data: Data partitioning and secondary indexes in the Hadoop ecosystem. *Information Systems*.

[17] Hao, C., & Ying, Q. (2011, October). Research of Cloud Computing based on the Hadoop platform. In *Computational and Information Sciences (ICCIS), 2011 International Conference on* (pp. 181-184). IEEE

[18] Vijayakumari, R., Kirankumar, R., & Rao, K. G. (2014). Comparative analysis of Google File System and Hadoop Distributed File System. *ICETS-International Journal of Advanced Trends in Computer Science and Engineering*, *3*(1), 553-558.

[19] Claudia Big Data Processing, 2013/14 Lecture 5: GFS & HDFS - distributed file systems Claudia Hauff (*Web Information Systems*).

[20] Daphalapurkar, A., Shimpi, M., & Newalkar, P. Mapreduce & Comparison of HDFS and GFS.

[21] The Apache HBase™ Reference guide, *http://hbase.apache.org/0.94/book.html*

[22] Centralized cache management in HDFS, *http://hadoop.apache.org/docs/current/hadoop-project-dist/hadoop-hdfs/CentralizedCacheManagement.html*

[23] Zhang, J., Wu, G., Hu, X., & Wu, X. (2012, September). A distributed cache for hadoop distributed file system in real-time cloud services. In *Grid Computing (GRID), 2012 ACM/IEEE 13th International Conference on* (pp. 12-21). IEEE.

[24] Yoon, S. D., Jung, I. Y., Kim, K. H., & Jeong, C. S. (2013, November). Improving HDFS performance using local caching system. In *Future Generation Communication Technology (FGCT), 2013 Second International Conference on*(pp. 153-156). IEEE.