Research Journal of Information Technology

ISSN 1815-7432

Academic Journals Inc.

www.academicjournals.com

OPEN ACCESS Research Journal of Information Technology

ISSN 1815-7432
DOI: 10.3923/rjit.2016.66.74

Research Article
Analyzing Google File System and Hadoop Distributed File System

Nader Gemayel

Department of Compuer Science, Notre Dame University-Louaize, P.O. Box 72, Zouk Mosbeh, Zouk Mikayel, Lebanon

Abstract

A comparative analysis study between Google file system and Hadoop distributed file system was conducted in this study. Using comarision techniques for architecture and development of GFS and HDFS, allows us use to deduce that both GFS and HDFS are considered two of the most used distributed file systems for dealing with huge clusters where big data lives. This study will help understand the architecture and highlight the common features and differences between GFS and HDFS.

Key words: Bigdata, GFS, HDFS, Hadoop, bigtable, HBase, MapReduce, analysis

Received: March 05, 2016 Accepted: June 21, 2016 Published: September 15, 2016

Citation: Nader Gemayel, 2016. Analyzing Google file system and Hadoop distributed file system. Res. J. Inform. Technol., 8: 66-74.

Corresponding Author: Nader Gemayel, Department of Computer Science, Notre Dame University-Louaize, P.O. Box 72, Zouk Mosbeh, Zouk Mikayel, Lebanon

Competing Interest: The author has declared that no competing interest exists.

Data Availability: All relevant data are within the paper and its supporting information files.

INTRODUCTION

Big data is a term used to describe huge volume of data, generated by digital process and media exchange all over the world. Google file system and Hadoop distributed file system were developed and implemented to handle huge amount of data. Big data challenges such as velocity, variety, volume and complexity were taken into consideration when GFS and HDFS were developed. Google came up first with the design of GFS and published it in white papers, then after Apache open-source developed Hadoop based on Google's white papers. How GFS and HDFS work and manage data flow will BBE explain. For both file systems, a comparison was made on many levels and criteria. The GFS and HDF'S architecture including database engines, components, data flow and many other creterias were tackled and discussed.

MAIN GOAL OF GFS AND HDFS

The HDFS and GFS were built to support large files coming from various sources and in a variety of formats. Huge data storage size (Peta bytes) are distributed across thousands of disks attached to commodity hardware. Both HDFS and GFS are designed for data-intensive computing and not for normal end-users[1]. Data-intensive computing is a class of parallel computing used to process and analyze large amount of data referred to as big data. Data reliability is achieved through distribution architecture, even when failures occur within chunk servers, master or network partitions. There is no limit to the cluster that you can have. The size can be increased anytime as per the need.

GOOGLE FILE SYSTEM

GFS architecture and components: The GFS is composed of clusters. A cluster is a set of networked computers. Figure 1 shows that, GFS clusters contain three types of interdependent entities which are: Client, master and chunk server. Clients could be: Computers or applications manipulating existing files or creating new files on the system. The master server is the orchestrator or manager of the cluster system that maintain the operation log. Operation log keeps track of the activities made by the master itself which helps reducing the service interruptions to a minimum level. At startup, master server retrieves information about contents and inventories from chunk servers. Then after, the master server keeps tracks of the location of the chunks with the cluster. The GFS architecture keeps the messages that the master server sends and receives very small. The master server itself doesn't handle file data at all, this is done by chunk servers. Chunk servers are the core engine of the GFS. They store file chunks of 64 MB size. Chunk servers coordinate with the master server and send requested chunks to clients directly.

GFS replicas: The GFS has two replicas: Primary and secondary replicas. A primary replica is the data chunk that a chunk server sends to a client. Secondary replicas serve as backups on other chunk servers. The master server decides which chunks act as primary or secondary. If the client makes changes to the data in the chunk, then the master server lets the chunk servers with secondary replicas, know they have to copy the new chunk off the primary chunk server to stay in its current state.

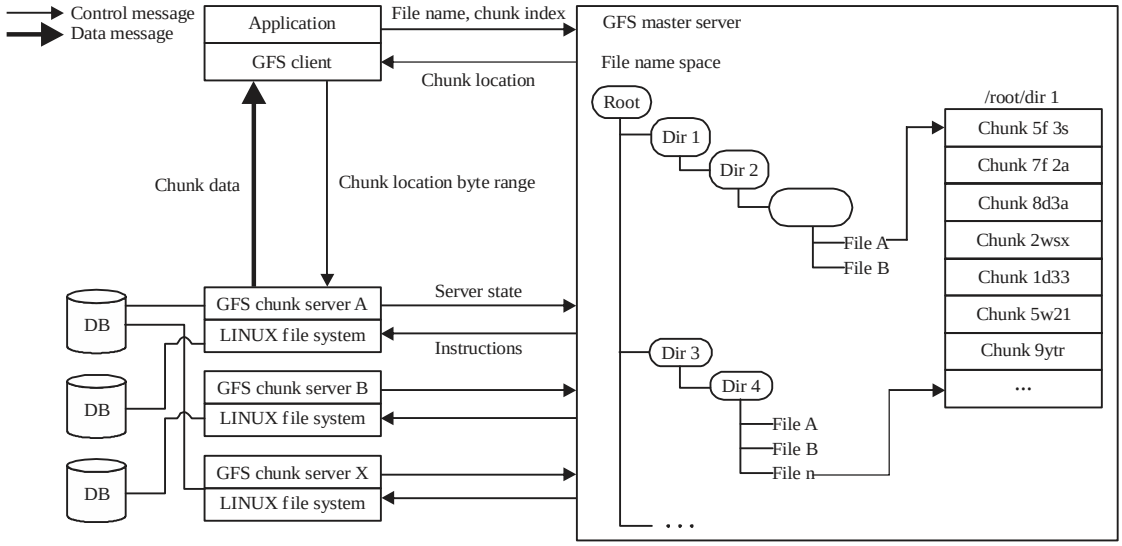

Fig. 1: Google file system architecture

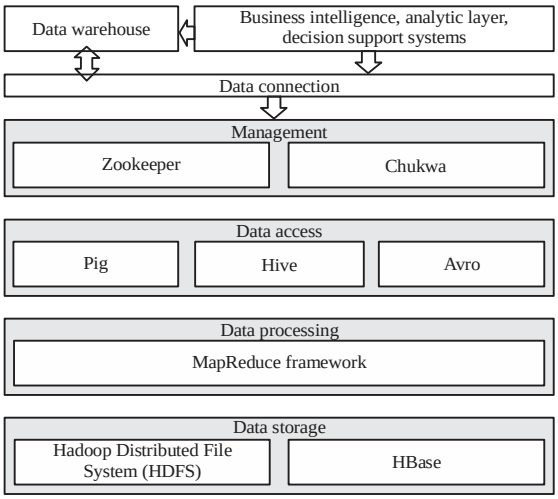

Fig. 2: Hadoop components

MapReduce: MapReduce is a programming model developed by Google and used by both GFS and HDFS. Based on Google MapReduce white paper, Apache adopted and developed its own MapReduce model with some minor differences. The primary role of MapReduce is to provide an infrastructure that allows development and execution of large-scale data processing jobs. Therefore, MapReduce exploits the processing capacity provided by computing clusters while, at the same time offering a programming model that simplifies the development of such distributed applications. MapReduce make the decomposition of tasks and integration of results. It provides job trackers and task trackers.

MapReduce is a programming model Google has used successfully to process big data. A map function extracts intelligence from raw data and a reduce function aggregates the data output by map. MapReduce needs a distributed file system and an engine that can distribute, coordinate, monitor and gather the results.

The HDFS is a master and slaver framework which contains nodes and NameNode. The NameNode is a center server that manages the namespace in the file system. The data node manages the data stored in it.

HADOOP DISTRIBUTED FILE SYSTEM

First of all, it should by clearly state that Hadoop has Google origins. Based on three white papers published by Google, which are: "Google file system[2]", "MapReduce: Simplified data processing on large clusters[3]" and "Bigtable: A distributed storage system for structured data[4]", Apache developed Apache HDFS, Apache MapReduce and Apache HBase, respectively. Almost 95% of the architecture described in these three white papers is implemented in Apache projects with some minor differences. Google released these white papers with no code. So, it was up to engineers and scientists at Apache to design and implement the architecture.

Hadoop components: Hadoop has the following components, as shown also in Fig. 2.

Zookeeper: A centralized service for maintaining configuration information, naming, providing distributed synchronization and providing group services.

Chukwa: An open source data collection system for monitoring large distributed systems. Chukwa is built on top of the Hadoop Distributed File System (HDFS) and MapReduce framework and inherits Hadoop's scalability and robustness.

Pig: A platform for analyzing large data sets that consists of a high-level language for expressing data analysis programs, coupled with infrastructure for evaluating these programs.

Hive: The Apache Hive data warehouse software facilitates querying and managing large datasets residing in distributed storage. Hive provides a mechanism to project structure onto this data and query the data using a SQL-like language called HiveQL.

Avro: Apache Avro is a data serialization system. It has rich data structures. A compact, fast and binary data format. It acts as a container file, to store persistent data. It also provides simple integration with dynamic languages.

MapReduce: Hadoop MapReduce has the same architecture and functionality of Google MapReduce but the difference is that Hadoop MapReduce was written in Java and GF'S MapReduce was written in c++.

Moreover, not to forget that Google's original version of MapReduce work only with GFS file system but Hadoop's version can work with many file systems since it was adopted by Apache open-source project hence used in many architectures.

HBase : Apache HBase is the Hadoop database, a distributed scalable and big data store

HDFS : Hadoop file system is a core component in the Hadoop architecture. The HDFS sits in the data storage layer in Hadoop. The HDFS and HBase will be explained in more details in the coming sections

Underlying file system: The HDFS is the distributed file system of Hadoop. What HDFS does is to create an abstract layer over an underlying existing file systems running on the machine. Underlying file systems might be ext3, ext4 or xfs.

Hadoop architechture: Since Hadoop comes from Google white papers, it has the same master/slave architecture but in different implementation. Hence, all processes or services in the Hadoop file system are classified as slave or master.

MasterNode-HDFS nodes

NameNode: Hadoop has only one NameNode (master node of the file system) which is a single point of availability failure, so if it goes down the DataNode will lose control for blocks.

Every slave machine will run a DataNode daemon and also run a TaskTracker daemon for MapReduce. NameNode stores the metadata of files saved on DataNodes. It is responsible for the namespace of the filesystem.

Hadoop 2.0 has also active/passive architecture for the NameNode. When active NameNode fails, a passive NameNode takes place within few seconds. This passive secondary NameNode is not really a good solution and does not really act as redundant to the primary NameNode, thus it is not a high availability to the primary NameNode.

The primary NameNode, which keeps all the filesystem metadata in RAM has no capability to persist this metadata onto the disk.

Once the NameNode crashes, all the data in RAM are lost. What the secondary NameNode actually does is to contact the primary NameNode every 1 h, copy the metadata in RAM, reshuffles and merges it into a clean file called a checkpoint. The checkpoint file is then written or updated on the secondary NameNode.

JobTracker: JobTracker receives job request from client and manages MapReduce jobs. It monitors and detects failures in task allocation. Hadoop cluster has one JobTracker only.

Slave node-MapReduce nodes

DataNode: DataNodes are the hardware machine running in Hadoop cluster. Usually built from inexpensive commodity, they are used to store data blocks and send them to clients. DataNodes report periodically the list of data blocks to the primary NameNode.

TaskTracker: TaskTrackers are Java-based virtual machine developed to run tasks allocated by the JobTracker in MapReduce. Usually TaskTrackers run better on the same node of the DataNode, which maximizes data bandwidth. If the above best-practice is not available, TaskTrackers are placed on different nodes within the same rack in cluster. In worst cases, if the alternative scenario is not also available, TaskTracker is placed on a different rack within the cluster.

COMPARISION BETWEEN GFS AND HDFS

Scalability: Both HDFS and GFS are considered as cluster based architecture. Each file system runs over machines built from commodity hardware. Each cluster may consist of thousands of nodes with huge data size storage.

Implementation: Since GFS is proprietary file system and exclusive to Google only, it can not be used by any other company.

In the other part, HDFS based on Apache Hadoop open-source project can be deployed and used by any company willing to manage and process big data.

Yahoo! might be the most famous example where clusters are managed by Hadoop with HDFS file system inside. Yahoo! has more than 100,000 CPU in 40,000 computers running Hadoop. Their biggest cluster contains around 4500 nodes.

Facebook uses Hadoop to store copies of internal logs and dimension data sources and uses it as a source for reporting/analytics and machine learning. They currently have two major clusters:

- A 1100-machine cluster with 8800 cores and about 12 Petabytes raw storage
- A 300-machine cluster with 2400 cores and about 3 Petabytes raw storage

EBay uses Apache on 532-machine cluster with Apache HBase for search optimization and research.

Twitter, LinkedIn, Adobe, A9.com (Amazon) and many other websites use Hadoop to store and process data logs, batch jobs, processes for internal usage and structured data storage on hundreds of clusters with thousands of nodes each.

File serving: In GFS, files are divided into units called chunks of fixed size. Chunk size is 64 MB and can be stored on different nodes in cluster for load balancing and performance needs. In Hadoop, HDFS file system divides the files into units called blocks of 128 MB in size[5]. Block size can be adjustable based on the size of data.

Internal communication: Communication between chucks and clusters within GFS is made through TCP connections. For data transfer, pipelining is used over TCP connections. The same method is in HDFS, but Remote Procedure Call (RPC) are used to conduct external communication between clusters and blocks.

Cache management: In GFS, cache metadata are saved in client memory. Chunk server does not need cache file data. Linux system running on the chunk server caches frequently accessed data in memory.

The HDFS has "DistributedCache". DistributedCache is facility provided by the MapReduce to distribute application-specific, large, read-only files efficiently. It also caches files such as text, archives (zip, tar, tgz and tar.gz) and jars needed by applications.

DistributedCache files can be private or public, that determines how they can be shared on the slave nodes.

"Private" DistributedCache files are cached in a local directory private to the user whose jobs need these files.

"Public" DistributedCache files are cached in a global directory and the file access is setup in sucha way that they are publicly visible to all users.

Files protection and permission: Suitebriar-Google partner-mentions in its security analysis research that GFS splits files up and stores it in multiple pieces on multiple machines. File names have random names and are not human readable. Files are obfuscated through algorithms that change constantly. The HDFS implements POSIX-like mode permission for files and directories. All files and directories are associated with an owner and a group with separate permissions for users who are owners, for users that are members of the group and for all other users.

Replication strategy: The GFS has two replicas: Primary replicas and secondary replicas.

A primary replica is the data chunk that a chunk server sends to a client.

Secondary replicas serve as backups on other chunk servers. User can specify the number of replicas to be maintained.

The HDFS has an automatic replication rack based system. By default two copies of each block are stored by different DataNodes in the same rack and a third copy is stored on a DataNode on a different rack.

File namespace:

- In GFS, files are organized hierarchically in directories and identified by path names
- The GFS is exclusively for Google only
- The HDFS supports a traditional hierarchical file organization
- Users or application can create directories to store files inside
- The HDFS also supports third-party file systems such as CloudStore and Amazon Simple Storage Service (S3)

Filesystem database: In this section, highlight one core component in the architecture of GFS and Hadoop; the database engine.

The GFS has bigtable database. Bigtable is a proprietary database developed by Google using c++.

Based on "Bigtable" study white papers, Apache developed its own database called HBase in Hadoop open-source project[6]. The HBase is built with Java language. The major common features between bigtable and HBase.

Big data: Ability to handle big amounts of data in a scalable manner.

No-SQL: Non relational database and not SQL based. The HBase is column-oriented database and bigtable is three-dimensional mapping database; it maps row key, column key and timestamp into one arbitrary byte array, hence the naming.

Atomicity: They both have atomic transactions in read, write and update.

Access control: Bigtable enforces access control on column family level with three aspects: Authentication, authorization and audit. This feature was not enabled in HBase until May, 2014.

Cell versioning and custom timestamps are found in HBase and bigtable using adjustable timestamps.

Batch writes: Batch write and batch table operations plus row filtering when scanning rows.

Block cache: Bigtable and HBase provide block cache. When reading blocks from storage files are cached internally.

Table 1: Bigtable anf HBase comparison

Feature/criteria	Bigtable	HBase
Software model	Proprietary/closed-source	Open-source
Framework used	c++	Java
Second log	Second log that can be used when the first log is not functioning fast	N/A
Locality group	Bigtable groups multiple column families into one group	HBase handles each column family separately
Memory mapping	Allows memory mapping of storage file directly into memory	N/A
File format	SSTable	HFile
Client script	Allows client script execution. Sawzall is used to enable users to process stored data	N/A
Working environment	Bigtable works only on GFS	HBase can work on many other file systems as long as proxy or driver class run on top of it
Block support and block compression	Block compression in Bigtable is based on BMDiff and Zippy two step process	GZip format with intention to use BMDiff and Zippy Till present, this issue is still under study by Apache
Client isolation	Bigtable keeps the data served for clients isolated from each others	N/A
Coprocessors	Coprocessors acts as high-level call interface for clients Very flexible model for developing distributed services Examples: Automatic scaling, load balancing and request routing for applications	This feature was initially developed by Google within its research on bigtable GFS before HBase acquired it November, 2011
Data verification	Bigtable uses CRC checksums to make sure that data was written safely	Technically HBase doesn't have this feature but was developed it with HDFS algorithm

In Table 1, key differences between Apache's HBase and Google's bigtable are listed. Some are still under development by Apache due to the fact that HBase came after and was built according bigtable specs.

DATA FLOW INPUT AND OUTPUT

GFS read I/O:

- Read requests are sent by clients to master in order to find out where a particular file on the system is stored
- Master server replies back with the location for the chunk server acting as the primary replica holding the chunk
- The master server provides a lease to the primary replica for the desired chuck
- If the lease is not held by any replica, the master server defines a chuck as primary and chooses the closest chunkserver to client. That chunkserver becomes the primary
- Finally, the client contacts the desired chunkserver directly, which sends the data to the client

GFS write I/O:

- The client sends a request to the master server to allocate the chunkserver acting as the primary replica (Fig. 3)
- The master sends to the client the location of the chunkserver replicas and identifies the primary replica
- The client sends the write data to all the replicas chunk server's buffer, starting with the closest. Data sent through pipeline
- Once the replicas receive the data, the client tells the primary replica to begin the write function
- The primary replica writes the data to the appropriate chuck and then the same is done on the secondary replica
- The secondary replica completes the write function and reports back to the primary replica
- Finally, the primary replica sends the confirmation to the client

HDFS read I/O:

- Client asks the NameNode about block's location
- NameNode has metadata for all blocks location. It sends blocks' location back to the client
- Client seeks and retrieves the blocks directly from DataNode where the blocks are placed

HDFS write I/O:

- The client sends a block write request to the NameNode (Fig. 4)
- The NameNode responds back by telling on which DataNodes the file's blocks should be written
- Directly, HDFS client contacts the first DataNode over TCP and sends "Ready" command. The first DataNode by its turn sends it to the second DataNode and the same process continues for the third DataNode
- "Ready" command is sent from the third DataNode to the second one and finally to the first DataNode which delivers it to the client telling all DataNodes are ready for the write order

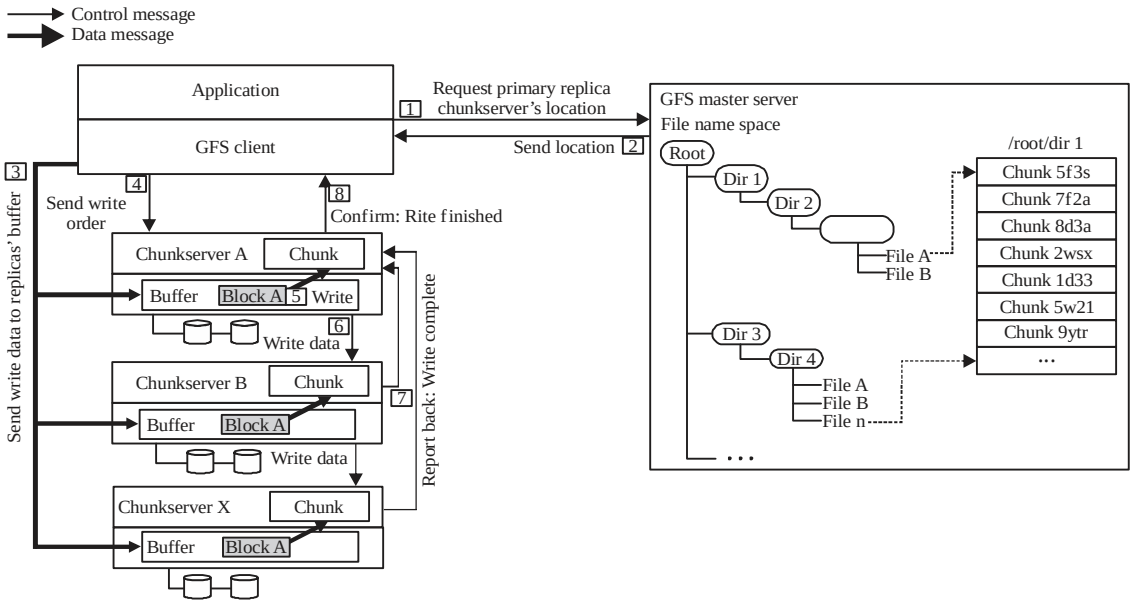

Fig. 3: GFS write I/O

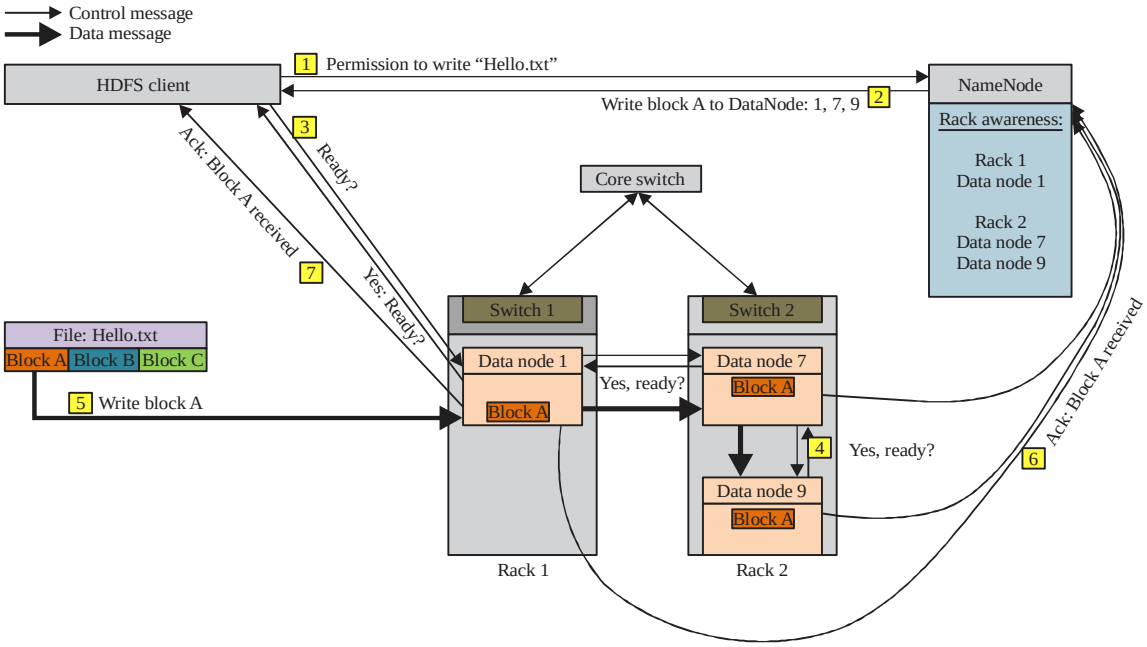

Fig. 4: HDFS write I/O

- The TCP pipeline is now ready to transfer the data block
- The client sends the first block wishing to write directly to the first DataNode, then the second and finally third DataNode
- All DataNodes update the NameNode about the written block
- First DataNode tells the client that file's block was written successfully
- Then after, the client repeats the same scenario for the rest of data blocks

MATERIALS AND RESULTS

Results of GFS and HDFS comparison can be resumed in the Table 2.

Table 2: Summary comparison

Feature/criteria	Google file system	Hadoop distributed file system
Main goal	Support large files	Support large files
	Variety of file formats	Variety of file formats
	Huge data storage size (Peta bytes)	Huge data storage size (Peta bytes)
	Data are kept forever	Data are kept forever
Design goal	Data-intensive computing	Data-intensive computing
	Not for normal end-users	Not for normal end-users
Underlying file systems	Only GFS file systems	Underlying file systems might be ext3, ext4 or xfs
Scalability	Cluster based architecture	Cluster based architecture
	Clusters may contains thousands of nodes	Clusters may contains thousands of nodes
Implementations	The GFS is proprietary file system	The HDFS based on Apache Hadoop open-source project used
	Can't be used by any other company	in Facebook, Twitter, LinkedIn, Adobe, A9.com (Amazon)
File serving	Chunk size is 64 MB	Block size is 128 MB. Can be adjustable
Data flow (I/O)	Master server and chunk server	NameNode and DataNode
Internal connections	The TCP connections	The TCP connections
	For data transfer, pipelining is used over TCP connections	Remote Procedure Call (RPC) is used to conduct external communication between clusters and blocks
Cache management	Cache metadata are saved in client's memory	The HDFS has "distributedcache"
	Chunk servers don't need cache file data	DistributedCache files can be private or public
	Linux system caches frequently accessed data in memory	
Files protection and permission	The GFS splits files up and stores it in multiple pieces on multiple machines	The HDFS implements POSIX-like mode permission for files and directories
Replication Strategy	The GFS has two replicas:	The HDFS has an automatic replication rack based system
	Primary replicas and secondary replicas	By default two copies of each block are stored
File system namespace	Files are organized hierarchically in directories and identified by path names.	The HDFS supports a traditional hierarchical file organization. Users or application can create directories to store files inside.
	The GFS is exclusively for Google only	The HDFS also supports third-party file systems such as CloudStore and Amazon Simple Storage Service (S3)
Database	Bigtable	HBase

DISCUSSION

Table 2 summarized the differences and commonalities between GFS and HDFS. The GFS and HDFS share many common features, but are also different in some ways. Google was the leader in big data management and design before Apache came up with their open-source project.

Large companies like facebook and Ebay relies on HDFS to manage their huge amount of data. Apache is still improving HDFS hence, becoming one of the biggest open-source projects ever developed.

CONCLUSION

Big data is increasing frequently hence becoming one of the major trends and challenges in information technology and dealing with it requires tailored-made file systems and hardware architecture, built on robust framework and platforms.

Distributed file system are still in their early phase evolution.

Coda, XtreemFS, AFS, lustre, ceph and many more are good examples of newly developed DFS.

It's up to the company's data engineers to choose and implement DFS based on their needs or to design and develop their own architecture as Google did.

ACKNOWLEDGMENT

I would like to express my special thanks and appreciation to my teacher Dr. Jacques Abou Abdo as well as my senior at Banque Libano-Francaise who provided me support and motivation towards accomplishing this research study.

REFERENCES

1. Kouzes, R.T., G.A. Anderson, S.T. Elbert, I. Gorton and D.K. Gracio, 2009. The changing paradigm of data-intensive computing. Computer, 42: 26-34.
2. Ghemawat, S., H. Gobioff and S.T. Leung, 2003. The google file system. Proceedign of the 19th ACM Symposium on Operating Systems Principles, October 19-22, 2003, ACM, Lake George, NY., pp: 29-43.

3. Dean, J. and S. Ghemawat, 2004. MapReduce: Simplified data processing on large clusters. Proceedings of the 6th Symposium on Operating Systems Design and Implementation, December 6-8, 2004, San Francisco, CA., USA., pp: 137-150.
4. Chang, F., J. Dean, S. Ghemawat, W.C. Hsieh and D.A. Wallach *et al*., 2008. Bigtable: A distributed storage system for structured data. ACM Trans. Comput. Syst., Vol. 26, No. 2. 10.1145/1365815.1365816.
5. Shafer, J., S. Rixner and A.L. Cox, 2010. The hadoop distributed filesystem: Balancing portability and performance. Proceedings of the IEEE International Symposium on Performance Analysis of Systems and Software, March 28-30, 2010, White Plains, NY., pp: 122-133.
6. Vora, M.N., 2011. Hadoop-HBase for large-scale data. Proceedings of the International Conference on Computer Science and Network Technology, Volume 1, December 24-26, 2011, Harbin, pp: 601-605.