International Journal of Innovations & Advancement in Computer Science
IJIACS
ISSN 2347 – 8616
Volume 4, Special Issue
March 2015

Academic Science

Google File System and Hadoop Distributed File System- An Analogy

Dr.A.P.Mittal[1] NSIT, New Delhi
Dr.Vanita Jain[2] BVCOE, New Delhi
Tanuj Ahuja[3] BVCOE, New Delhi

ABSTRACT
Big Data has indeed been the word which IT Industry is talking about lately. With advancement of automation and data being processed in real time, it has now become a necessity for companies to look forward to sustainable solutions to store their huge datasets and compute valuable information out of it. High performance computing heavily relies on distributed environments to process large chunks on data. With ever-increasing data, a reliable and easy to use storage solution has become a major concern for computing. Distributed File System tries to address this issue and provides means to efficiently store and process these huge datasets.
This paper presents and compares two common distributed processing frameworks involved in dealing with storage of large amounts of data- Google File System (More commonly now known as 'Colossus') and Hadoop Distributed File System. Also, it gives an insight to the MAPREDUCE framework common to both GFS and HDFS.
KEYWORDS
GFS, HDFS, MapReduce, Distributed Storage, Parallel Processing

1. INTRODUCTION

Data is primary asset of any Organization and if the organization knows how to exploit it, they'll surely benefit from it. Over the last decade, the data which has been generated from varying sources such as web searches, call direct records, social media chats, artificial intelligence, email and other information streams is enormous. Enterprise and organizations now seek to leverage this tremendous amount of data in order to gain a better insight into Business processes. The primary goal is to analyze the scattered data and predict future trends and business intelligence solutions which would benefit the enterprise and client all together [1, 2].
The four step process followed by Companies to exploit the data can be stated as-
i) ACQUIRING the Unstructured Data in form of web clicks, chats, call detail record etc.
ii) ORGANIZING these chunks of data in traditional structured form by means of parallel computing on distributed storage.
iii) ANALYSING the trends and patterns available from data.
iv) DECIDING upon the conclusions and steps to be taken to utilize it in full sense [3,4].

A Distributed File System provides a means of permanent storage of set of data, typically in form of files. This storage consists of several resources and features wherein; clients can create, delete, read or write files. Clients and storage resources are dispersed over a network, unlike the prior local file system. Although files are stored on different resources, it gives an impression to the user to be present at same geographical location. Sharing of files is done in a hierarchical way. The factors of transparency, fault tolerance and scalability are considered to be of utmost importance by a Distributed File System [5].

With ever-growing data being generated in form of web searches, emails, advertisements, clicks and other sources [6], Google decided to develop its own Distributed File System with a motive to improve its data handling capacity. The primary reason behind the birth of Google File System (GFS) was to provide fault tolerance and resilience [7].
Following the footsteps, few years later, a similar implementation provided an open source framework which is used for distributed storage and processing and is called Hadoop. Being open-source, it gained popularity with big companies like Facebook, Amazon [8] using it to address their data issues.
This paper is structured as; Section 2 and Section 3 describes about the Architecture and Overview of Google File System and Hadoop's implementation of distributed storage- Hadoop Distributed File System respectively. Section 4 discusses about the MapReduce technique common to both. Section 5 provides a comparative analysis of both frameworks followed by an insight of future trends and scope.

2. GOOGLE FILE SYSTEM

As a consequence of the services provided by Google, it faces the requirements to effectively manage large amounts of data involving web text/image searches, maps, YouTube etc. Considering the fact that the available data is huge and vulnerable, Google tried to create its own file storage structure and came up with the concept of Google File System(GFS) in 2003[7]. The basic considerations which forced Google to switch to a new file system included constant monitoring, fault tolerance as well as error detection and automatic recovery.
Having the clear aims of a prospective file system in detail, Google has opted not to use an existing distributed file system such as Network File System(NFS) [9] or Andrew File System(AFS) [10] but instead decided to develop a new file system customized fully to suite Google's needs.

A major difference introduced by Google was the fact that GFS does not provide a **P**ortable **O**perating **S**ystem **I**nterface (POSIX) [11] interface, which means that GFS also does not integrate with the Linux Virtual File System (VFS) layer. Instead, GFS implements a proprietary interface.

The primary aims of GFS involved managing distribution, coping with increasing danger of hardware faults (disk faults, machine faults and network faults), ensuring data safety and providing a mechanism for scalability. These issues were rigorously addressed by the Google File System until lately, when Google came up with an optimized successor of GFS, commonly called **Colossus** [12].

Colossus tried to improve upon the performance and issues involved in GFS. These will be discussed in later sections of the paper.

2.1 Architecture

GFS follows a master-chunkserver relationship. It consists of a single master and a number of chunkservers. Both the master and the chunkservers can be accessed by multiple clients. Files are divided into fixed size chunks, with default chunk size being equal to 64MB. A chunk handle which is usually a 64 bit pointer, is used by master to handle each chunk [7]. The handle is globally unique and immutable (write once, read anywhere). Each chunk is replicated on multiple chunkservers for

reliability. By default, the number of replicas created is three but it can be changed accordingly. The general architecture of Google File System is shown Figure 1 [7].

2.1.1 Master

One of the primary roles of Master is to maintain all Metadata. This includes managing file and chunk namespaces, keeping track of location of each chunk's replica, mapping from files to chunks and accessing control information [13]. Storing metadata in master provides simplicity and flexibility compared to storing it individually for each chunk. A general norm which is followed by the master is that it allows less than 64 bytes of metadata being maintained for each 64 MB chunk.

Besides, master is also responsible for chunk management and chunk migration between chunkservers. Also, it supplies garbage collection mechanism to avoid fragmentation. The master periodically gives instructions to each chunckserver and gathers information about its state also.

2.1.2 Client

The role of the client is to ask the master as to which chunkserver to contact. Using file name and byte offset, it creates a chunk index and sends it to master. Master on the other hand acknowledges the current and few future requests. This ensures minimum future interaction between client and master.

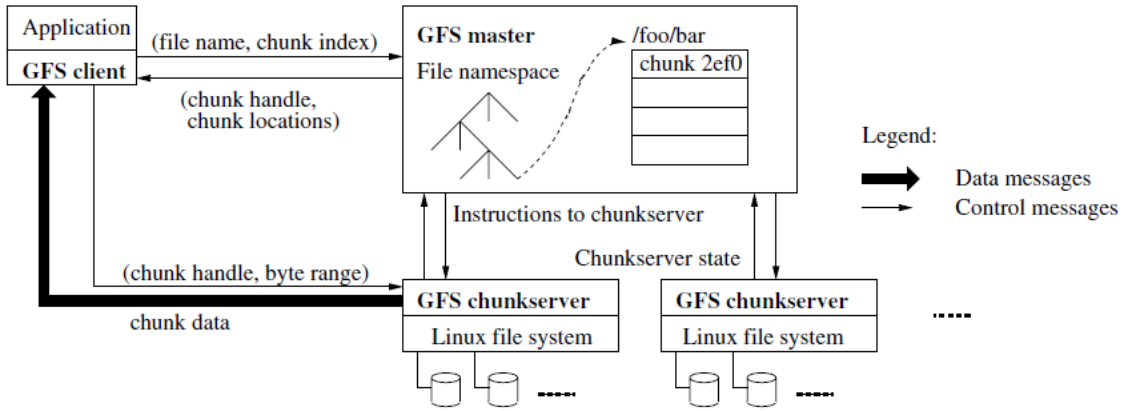

Figure 1: GFS Architecture

2.2 Snapshot

A Snapshot is an integral function involved in functioning of Google File System, which ensures consistency-control. It helps a user to create a copy of a file or a directory instantaneously, without hindering the on-going mutations. It is more commonly used while creating checkpoints of the current state in order to perform experimentation on data which can later be committed or rolled back.

2.3 Data Integrity

Given the fact that a typical GFS cluster consists of thousands of machines, it experiences disk failures time and again, which ultimately causes corruption or loss of data reads and data writes. Google considered this to be a norm rather than an exception. To avoid this problem, each chunkserver maintains its own copy of checksum. A checksum ensures integrity of chunkserver by

verifying it with the master. Each chunk broken up into 64KB block has a corresponding checksum of 32 bits. Similar to metadata, checksums are also stored with logging, independent from user data.

2.4 Garbage Collection

Instead of immediately reclaiming the physical storage space after a file or a chunk is deleted, GFS implies a lazy action policy of Garbage Collection. This approach ensures that system is much simpler and more reliable.

2.5 Implication

GFS relies on appends rather than overwrites. It believes in the strategy of checkpointing in order to fully utilize its fault tolerance capabilities. By differentiating file system control (master) from data transfer (chunkserver and clients), it ensures that master's involvement in common operations is minimized, resulting in faster execution.

2.6 Successor of GFS- Colossus

Google lately changed its focus to automation of data. This led to an entire architectural change in existing distributed storage and processing. This new automated type of storage was named 'Colossus' and is being used by Google presently. Colossus is the next generation cluster level file system with data being written to it using 'Reed Solomon error correction codes' which ultimately account for a 1.5 times redundancy compared to its predecessor. It is a client driven system with full access given to client to replicate and encode data.

3. HADOOP DISTRIBUTED FILE SYSTEM

Hadoop is an Open Source implementation of a large-scale batch processing system. It is a software framework which is used for distributed storage and also for distributed processing of data on clusters of commodity hardware [14].

Hadoop is currently being used for index web searches, email spam detection, recommendation engines, prediction in financial services, genome manipulation in life sciences, and for analysis of unstructured data such as log, text, and click stream [15].

It implements MapReduce framework initially introduced by Google. It provides Java based environment but custom codes in other languages can also be processed. Depending upon the complexity of the process and volume of data, parallel processing response time can vary from minutes to hours [16]. Besides fast processing, the major advantage Hadoop brings along is its massive scalability [17].

HDFS is a distributed file-system that stores data on commodity machines, providing very high aggregate bandwidth across the cluster.

Hadoop Distributed File System is designed to handle mammoth amount of data with streaming data access patterns across multiple machines [18]. It is a distributed, fault tolerant file system used for massive data clustered storage. Its work is to split the incoming data into blocks which are duplicated and distributed across several nodes on the Hadoop cluster[19,20].

3.1 Architecture

HDFS follows a master-slave relationship. It has a single 'namenode' and multiple 'datanodes'. The namenode stores the metadata and henceforth acts as an index which can be referred to gain knowledge about the location of data. The datanode simply contains the data.

The later versions of Hadoop have an additional node named 'secondarynode'. This node is an imitation of namenode stored at some location other than the namenode. Its purpose is to perform

periodic checkpoints. It periodically downloads the current namenode image and edits log files, joins them in new image and uploads new image back to the namenode [21].
Each file in HDFS appears as a large conterminous stream of bytes on which parallel map reduce programming is done. Figure 2 depicts the Hadoop architecture [1].

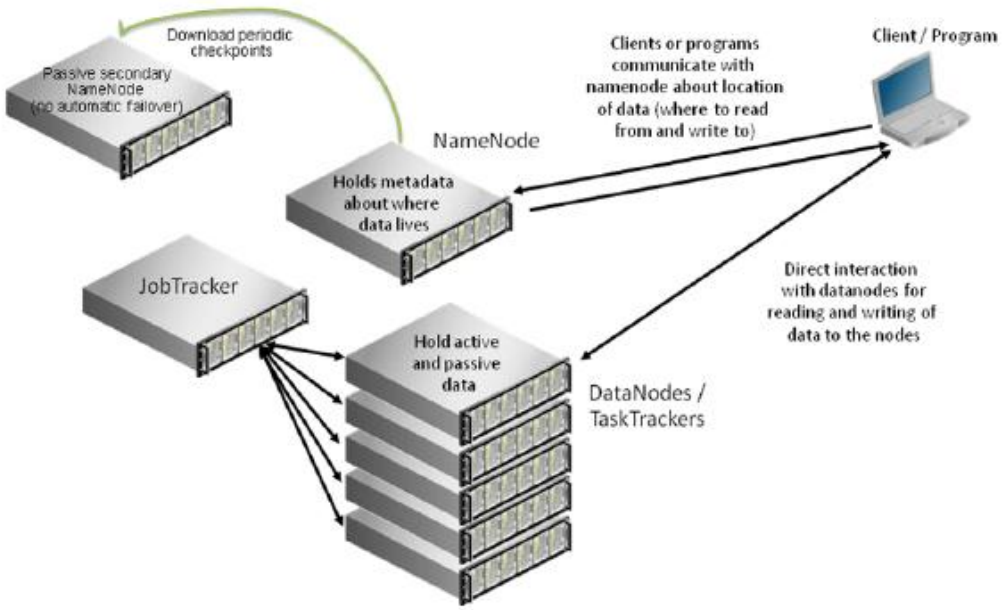

Figure 2: Hadoop Architecture

3.2 Naming

HDFS uses inodes (used to record attributes like permissions, access times and modification) to handle its namespaces in form of hierarchial files and directories. Namenode manages the metadata(data/index about data) and namespaces and is also responsible for mapping filename and file blocks on datanodes.

3.3 Cache consistency

HDFS follows a write only, read many model which ensures concurrency control as the client is not allowed to modify the file once created but is allowed to read it several times. In case of a network or server failure, data consistency is ensured by checksum, which is generated for each file block. The client can compare his computed checksum with the checksum of file block stored in HDFS in order to ensure a valid read.

3.4 Replication of Data

HDFS follows a default placement policy to replicate and distribute data blocks across several datanodes. By default, three copies of a block are created. This number can be varied by the user

whenever desired. It is ensured that no datanode holds more than one copy of the block and no rack holds more than two copies of same block. Namenode verifies that intended number of copies is placed at suitable locations and it takes action in case of over-replication or under-replication by balancing the amount of available disk space across the racks.

3.5 Load balancing

Utilization of server is ensured by computing the ratio of the space used to the total capacity of the server. This ratio is between a threshold range of (0, 1). A cluster is balanced if server usage for each datanode is no more than threshold value. Otherwise, it is unbalanced and copies are moved from it to other servers to ensure load balancing.

3.6 Fault Detection

Each datanode is assigned a namespace ID which is compared with those held by namenode, every time the server starts. If a mismatch occurs, datanodes are shut down, ensuring the data integrity is maintained. Datanodes also perform registration with namenode which helps in their recognition in case they start with different IP address or port. A heartbeat message is sent every three seconds to the namenode to confirm availability. A datanode is considered out-of-service if no heatbeat is received in a time span of ten minutes[22].

4. MAPREDUCE

Map Reduce is a strategy employed to process and monitor large data sets in a computer cluster or it can be defined as a programming model and an associated implementation which can be used to process and generate large data sets. It is used to automate a routine of two functions namely map() and reduce().The basic unit of information is a <key; value> pair.

MapReduce function was initially introduced by Google for processing their own large datasets. The abstraction was inspired by the Map and Reduce primitives present in Lisp and other functional languages [23]. It followed a master-worker relationship with master storing the state of the worker, that is, ‘idle’, ’in-progress’ and ‘completed’ and identity of the worker.
Map() function takes the input ,implementing abstract file indexing with built-in functions of mapping, shuffling, sorting, joining and maintaining data in the form of stack storage.
The **reduce()** function then take those sorted data sets and operate on them and finally produces the result.
A MapReduce cluster follows a master-slave architecture in which one master node manages a number of slave nodes pulling each other’s strings in order to produce centralized environment for extracting significant results. Figure 3 depicts the Map Reduce concept [24].
In Hadoop, the master node is called ‘JobTracker’ and the slave node is called ‘TaskTracker’. Hadoop uses this MapReduce job. This is done by first splitting the input data set into an even-sized data blocks. Then each data block is assigned to TaskTracker for processing and reducing them to form sorted sets of data[25]. In order to process huge and heterogeneous sets of big data, MapReduce provides a solution for parallel and distributed input output scheduling making it fault tolerant and scalable [26].

Job Tracker: This node is master, which creates the job and allocates it to the task trackers. It runs on the name node and hence tracks the stored data.
Task Tracker: The task tracker runs on the datanode. It executes the job assigned to it and reports the processing status to the job tracker [27,28].

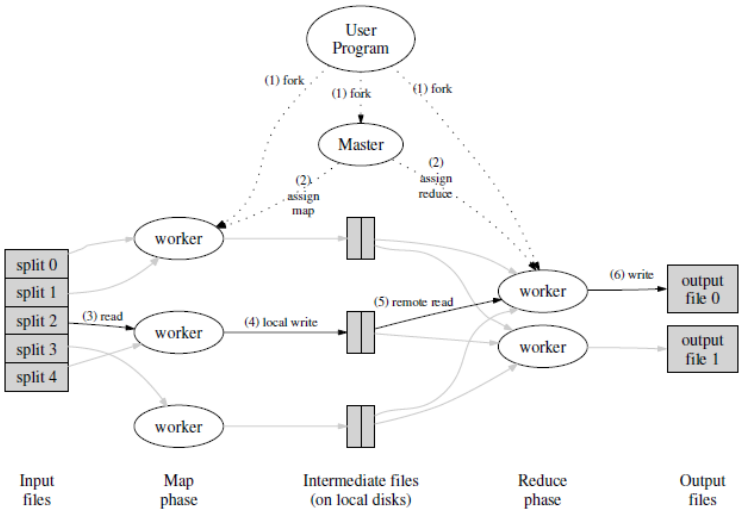

Figure 3: Map Reduce Concept

5. COMPARATIVE ANALYSIS

This section gives an insight to the working style, implementation, scalability criteria, fault tolerance and other parameters of the two systems under study, which are Google File System and Hadoop Distributed File System.

5.1 Implementation

GFS is written completely on Linux platform. The programming language opted by Google is C/C++. It is proprietary software written by Google for storage of their own data.
HDFS on the other hand is an integral part of Hadoop framework, written for cross-platform usage. The primary programming language followed is JAVA but implementations can be made using other languages as well. It is open source software developed mainly by Yahoo and open source community. The license is held by Apache[29].

5.2 Architecture

GFS is a centralized distributed file system. The naming convention involves master-chunkserver relationship wherein only one copy of master is maintained (In recent successor of GFS, that is COLOSSUS, more than one copy of master is maintained.) which stores metadata using a chunk handle. The architecture is such that it tries to minimize the interaction between master and client. By default, three replicas of each chunk are created on various racks, to ensure reliability. The chunk size is 64MB.
HDFS is also a centralized distributed file system. The naming convention followed is namenode-datanode relationship wherein one copy of namenode is maintained (secondary namenode is also maintained in case primary namenode fails) which is responsible for allocating tasks to various datanodes (slaves).

5.3 Fault Tolerance

GFS assumes fault tolerance to be a norm rather than an exception. The most basic technique it uses to avoid faults is fast recovery, which ensures no matter what, that both master and chunkserver are quickly restarted in case of a fault. Besides, Chunk Replication is used such that in case of failure of a chunk, another one from different rack can be used without affecting the overall execution. It also implements Master Replication which creates a 'Shadow' copy of the master. This copy assumes every successful change made to the master and acts exactly as an inactive master. In case the master falters, which usually doesn't happen, the control switches to the shadow copy.

Fault Tolerance concept is implemented by the Mappers and Reducers as well. If the worker fails, it is switched to initial idle state. If few map-tasks are completed on a failed machine, they are re-executed. Completed reduce-tasks are not re-executed because they are saved to a global file system. The master handling the workers writes periodic checkpoints so that in case of failure, last saved checkpoint state can be accessed.

In HDFS, Datanodes perform a registration with the namenode so that they can be recognized even if they start at different IP address. After this registration, datanodes send block report to the namenode, after a fixed interval of time, to provide the latter with the information about the blocks held. Every three seconds, datanodes send heartbeats to the namenode to confirm their availability. If no heartbeat is received in a time span of ten minutes, then datanode is considered to be out-of-service[30].

5.4 Map Reduce Type

Although Google initially started with the concept of Map() and Reduce() functions which involved providing <key, value> pairs and then processing on it, it lately changed it strategy to a more optimized one. Initially, some information was first gathered and then sent for processing to Mappers. This batch type of processing was replaced by more dynamic one. With introduction of successor of GFS, that is, Colossus, Google started with real-time processing of data with processing being carried out as soon as the data became available. This gave birth to the services of Google like Google Instant.

On the other hand, Hadoop still uses the conventional batch type processing which involves gathering and then feeding data for processing.

5.5 Security

Google datacenters are usually present at undisclosed locations with very few actually knowing about their geographical presence. Only authorized employees are allowed to access these datacenters. Electronic key access, patrol teams, CCTV coverage, accessed logs are few measures adopted by Google to ensure tight security[29].

HDFS follows a very simple and vulnerable model. Only file level security access is required to check the status. The security varies from one host operating system to other.

5.6 Metadata Handling

HDFS lacks the automatically shared metadata layer supported by Colossus (GFS). HDFS has a different but related concept of an HDFS Federation that allows multiple namespaces. Each namespace is powered by a different name node, to co-exist on the same set of data nodes. But the administrator has the right to ensure that the metadata in any given namespace doesn't exceed memory in the name node.

5.7 Data Integrity and Disk Usage

Reed-Solomon error codes that provide 1.5x redundancy help reduce disk usage while still protecting data integrity by using parity bits to protect against corruption. Facebook has been working on building this into HDFS Raid since 2010 , but it remains an open issue for HDFS. [12]

5.8 Geographic Redundancy

Colossu has the ability to replicate blocks across different clusters for geographic redundancy which is something that HDFS has yet to support. Note that this is entirely different from just having a single name node communicating to many data nodes in different geographic regions because for performance in HDFS clusters locality is assumed.

5.9 Software Ecosystem

Much of Google's innovation in the storage space, beyond its original Google File System, happens in the software ecosystem that treats GFS as a basic abstraction upon which other things are built. BigTable has provided distributed structured storage on top of GFS since before 2006 [7], and Spanner has unified all datacenters as one storage and computational unit since 2009-10 [12].

With Spanner, Google's storage system has grown very vast that it essentially treats an entire datacenter, rather than an individual machine, as the basic building block. It also automatically scale resources and replicas based on network bandwidth, usage patterns and other prevailing constraints. Across entire fleets of machines, auto scaling of resources happen.

HDFS and HBase, which are open source Apache projects are making good progress at becoming extremely useful to companies at non-Google scale. They provide the similar systems, but are definitely behind.

5 CONCLUSION

This paper gives an insight of the working process of two powerful data processing frameworks, namely Google File System (more commonly ‘Colossus’) and Hadoop Distributed File System. A comparative analysis was performed to examine the performance of both the systems.

It can be concluded that to effectively handle hard drive failures, power failures, router failures, network maintenance, bad memory, rack moves, misconfigurations, datacenter migrations, etc. across hundreds of thousands or millions of machines requires significantly better error monitoring, tooling, and auto-recovery, which is being provided in far better way for GFS, compared to a similar Hadoop cluster. This knowledge is something that will take developers on HDFS and Hadoop much longer to develop. Although efforts are being made by firms like Apache to come up with real-time monitoring systems (Spark [31]) which are more automated and efficient, the efforts still lag behind. And therefore, for the time being Colossus, the successor of GFS prevails its roots as the better framework for distributed storage.

REFERENCES

[1] Pedro lay and Jean-Pierre Dijcks, Leveraging Massively Parallel Processing in an Oracle Environment for Big Data Analytics, Oracle White Paper, November 2010.

[2] Ms. Vibhavari Chavan, Prof. Rajesh and N. Phursule, Survey Paper On Big Data, International Journal of Computer Science and Information Technologies, Volume 5, Issue 6 , pp. 7932-7939 , 2014.

[3] Mike Gualtieri and Noel Yuhanna, The Forrester Wave™: Big Data Hadoop Solutions, February 27, 2014 | Updated: February28, 2014

[4] Big Data: The Next Frontier for Innovation, Competition and Productivity : http://www.mckinsey.com/Insights/MGI/Research/Technology_and_Innovation/Big_data_The_next_frontier_for_innovation, June 2011.

[5] Hadoop and HDFS:Storage for next generation Data Management, White Paper, Cloudera, 2014.

[6] Tushar M.Chavan and S.P.Akarte, Opportunities and Challenges of Big Data in Economics Research and Enterprises, International Journal of Computer Science and Mobile Computing, Volume 3 , Issue. 4, April 2014.

[7] Sanjay Ghemawat, Howard Gobioff, and Shun-TakLeung, The Google File System, ACM Symposium on Operating Systems Principles, Lake George, NY, pp. 29 – 43, Oct 2003.

[8] SteveLohr, The Age of Big Data ,New York Times, Feb 11,2012. 'http://www.nytimes.com/2012/02/12/sunday-review/big-datas-impact-in-the-world.html'

[9] S. Shepler, B. Callaghan, D. Robinson, R. Thurlow, C. Beame, M. Eisler, D. Noveck,Network File System (NFS) version 4 Protocol, RFC 3530. , Network Appliance, Inc., April 2003.

[10] John Howard, Michael Kazar, Sherri Menees, David Nichols, Mahadev Satyanarayanan, Robert Sidebotham, and Michael West. Scale and performance in a distributed file system. ACM Transactions on Computer Systems, Volume 6, Issue 1, pp. 51–81, February 1988.

[11] IEEE, Information Technology Portable Operating System Interface (POSIX) Part 1: System Application Programming Interface (API), 1003.1-1990, 1990.
[12] Andrew Fikes, Principal Engineer,Storage Architecture and Challenges,
Faculty Summit, Google ; July 29, 2010.
[13] Jeffrey Dean and Sanjay Ghemawat, MapReduce: Simplied Data Processing on Large Clusters, Google Inc., 2008.
[14] Apache: Apache Hadoop, 'http://hadoop.apache.org'
[15] Arantxa Duque, Barrachina,and Aisling O'Driscoll, A big data methodology for categorizing technical support requests using Hadoop and Mahout, Journal of Big Data, Volume 01, Issue 01, pp01-11, 2014.

[16] Hadoop Distributed File System.[Online].Available:http://www.ibm.com/developer works/web/library/wa-introhdfs

[17] DunrenChe, MejdlSafran, and ZhiyongPeng, From Big Data to Big Data Mining: Challenges, Issues, and Opportunities, DASFAA Workshops 2013, LNCS 7827, pp. 1–15, 2013.

[18] J.Shafer, S.Rixner, and A.L.Cox, The Hadoop distributed file system: Balancing portability and performance", IEEEInternational Symposium on Performance Analysis of Systems & Software (ISP ASS), pp. 122-133, 2010.

[19] D. Borthakur. HDFS Architecture Guide. [Online] Available: 'hadoop.apache.org/docs/hdfs/current/hdfs_design.html'

[20] Apache HDFS. Available at 'http://hadoop.apache.org/hdfs'

[21] Puneet Singh Duggal and Sanchita Paul, Big Data Analysis: Challenges and Solutions ,International Conference on Cloud, Big Data and Trust, RGPV, India Nov 13-15, 2013.

[22] Benjamin Depardon, Ga□el Le Mahec, Cyril S_eguin. Analysis of Six Distributed File Systems.[Research Report] 2013, pp.44. <hal-00789086>

[23] Johannes Passing, The Google File System and its application in
MapReduce, Seminar Software Design, Hasso-Plattner-Institute for Software Engineering, 2007-2008
[24] Jeffrey Dean and Sanjay Ghemwat, MapReduce: Simplified data processing on large clusters, Communications of the ACM, Volume 51 , Issue 01, pp. 107–113, 2008.

[25]Suman Arora, Dr.Madhu Goel, Survey Paper on Scheduling in Hadoop, International Journal of Advanced Research in Computer Science and Software Engineering, Volume 4, Issue 5, pp. 812-815,May 2014.

[26] Jeffrey Dean and Sanjay Ghemwat, MapReduce:A Flexible Data Processing Tool, Communications of the ACM, Volume 53, Issue 1, pp 72-77, January 2010.

[27] Howard Karlo, Siddharth Suri,Sergei Vassilvitskiiz, A Model of Computation for MapReduce. Available at 'theory.stanford.edu'

[28] MapReduce tutorial. [Online]. Available:'http://hadoop.apache.org/common/docs/r0.20.2/mapred_tutorial.html'

[29] R.Vijayakumari, R.Kirankumar, K.GangadharaRao, Comparative analysis of Google File System and Hadoop Distributed File System, International Journal of Advanced Trends in Computer Science and Engineering, Vol. 3 , No.1, Pages : 553– 558 , 2014.

[30] Ameya Daphalapurkar, Manali Shimpi, Priyal Newalkar, Mapreduce& Comparison of HDFS and GFS, International Journal of Engineering and Computer Science, Volume 3, Issue 9, pp. 8321-8325,September, 2014.

[31] Matei Zaharia, Mosharaf Chowdhury, Michael J. Franklin, Scott Shenker, Ion Stoica , Spark: Cluster Computing with Working Sets ; University of California, Berkeley HotCloud 2010, June 2010.