ISSN 2278-3091

International Journal of Advanced Trends in Computer Science and Engineering, Vol. 3 , No.1, Pages : 553– 558 (2014)
Special Issue of ICETETS 2014 - Held on 24-25 February, 2014 in Malla Reddy Institute of Engineering and Technology, Secunderabad– 14, AP, India

Comparative analysis of Google File System and Hadoop Distributed File System

R.Vijayakumari, R.Kirankumar, K.Gangadhara Rao
Dept. of Computer Science, Krishna University, Machilipatnam, India, vijayakumari28@gmail.com
Dept. of Computer Science, Krishna University, Machilipatnam, India, kirankreddi@gmail.com
Dept. of Computer Science and Engg., Acharya Nagarjuna University, Guntur, India, kancherla123@gmail.com

Abstract – Cloud computing is a new technology which comes from distributed computing, parallel computing, grid computing and other computing technologies. In cloud computing, the data storage and computing are not in the local computer and server but in the amount of computers distributed in the internet. Several distributed file systems are used over the cloud because the cloud itself includes large numbers of commodity-grade servers, harnessed to deliver highly scalable and on-demand services. The main objective of this paper is to discuss about two distributed file systems Google File System (GFS) and Hadoop Distributed File System (HDFS) and compare them by making various use of parameters. Mapreduce is a functional programming model introduced by Google and is used by both GFS and HDFS. Parameters such as Design Goals, Processes, Fie management, Scalability, Protection, Security, cache management replication etc. are taken for comparison.

Keywords: Cloud computing, Google file system, Hadoop distributed file system, Mapreduce.

INTRODUCTION

Cloud computing is a specialized distributed computing paradigm; it differs from traditional ones in that 1) it is massively scalable, 2) can be encapsulated as an abstract entity that delivers different levels of services to customers outside the Cloud, 3) it is driven by economies of scale [1], and 4) the services can be dynamically configured (via virtualization or other approaches) and delivered on demand.

Google File System [2] is a proprietary distributed file system developed by Google and specially designed to provide efficient, reliable access to data using large clusters of commodity servers. Files are divided into chunks of 64 megabytes, and are usually appended to or read and only extremely rarely overwritten or shrunk. Compared with traditional file systems, GFS is designed and optimized to run on data centers to provide extremely high data throughputs, low latency and survive individual server failures. Inspired by GFS, the open source Hadoop Distributed File System (HDFS)[3] stores large files across multiple machines. It achieves reliability by replicating the data across multiple servers. Similarly to GFS, data is stored on multiple geo-diverse nodes. The file system is built from a cluster of data nodes, each of which serves blocks of data over the network using a block protocol specific to HDFS. In order to perform the certain operations in GFS and HDFS a programming model is required. GFS has its own programming model called Mapreduce. It is an open-source programming model developed by Google Inc. Apache adopted the ideas of Google Mapreduce and developed Hadoop Mapreduce.

In this paper comparison is made in terms of the features of two distributed file systems: Google File System (GFS) and Hadoop Distributed File System [4] which is an open-source implementation of Google file system [5].

GOOGLE FILE SYSTEM (GFS)

The Google file system is implemented to meet the rapidly growing demands of Google’s data processing needs. Google faces the requirements to manage large amounts of data – including but not being limited to the crawled web content to be processed by the indexing system. Relying on large numbers of comparable small servers [6], GFS is designed as a distributed file system to be run on clusters up to thousands of machines. In order ease the development of applications based on GFS, the file system provides a programming interface aimed at abstracting from these distribution and management aspects. Running on commodity hardware, GFS is not only challenged by managing distribution, it also has to cope with the increased danger of hardware faults. Consequently, one of the assumptions made in the design of GFS is to consider disk faults, machine faults as well as network faults as being the norm rather than the exception. Ensuring safety of data as well as being able to scale up to thousands of computers while managing multiple terabytes of data can thus be considered the key challenges faced by GFS. Having distilled the aims and non-aims of a prospective file system in detail, Google has opted not to use an existing distributed file system. Instead it decided to develop a new file system. GFS has been fully customized to suite Google's needs. This specialization allows the design of the file system to abstain from many compromises made by other file systems. As an example, a file system targeting general applicability is expected to be able to efficiently manage files with sizes ranging from very small (i.e. few bytes) to large (i.e. gigabyte to multi-terabyte). GFS, however, being targeted at a particular set of usage scenarios, is optimized for usage of large files only with space efficiency being of minor importance. Moreover, GFS files are commonly modified by appending data, whereas modifications at arbitrary file offsets are rare. The majority of files can thus, in sharp contrast to other file systems, be considered as being append-only or even immutable (write once, read many). Coming along with being optimized for large files and acting as the basis for large-volume data processing systems, the design of GFS has been optimized for large streaming reads and generally favors throughput over latency. GFS implements a proprietary interface applications can use.

ARCHITECTURE OF GFS

A GFS cluster consists of a single *master* and multiple *chunkservers* and is accessed by multiple *clients*, as shown in Figure 1. Each of these is typically a commodity Linux machine running a user-level server process. It is easy to run both a chunk server and a client on the same machine, as long as machine resources permit and the lower reliability caused by running possibly flaky application code is acceptable. Files are divided into fixed-size *chunks*. Each chunk is identified by an immutable and globally unique 64 bit *chunkhandle* assigned by the master at the time of chunk creation. Chunk servers store on local disks as Linux files and read or write chunk data specified by a chunk handle and byte range. For reliability, each chunk is replicated on multiple servers. By default, we store three replicas, though users can designate different replication levels for different regions of the file namespace. The master maintains all file system metadata. This includes the namespace, access control information, the mapping from files to chunks, and the current locations of chunks. It also controls system-wide activities such as chunklease management, garbage collection of orphaned chunks, and chunk migration between chunk servers. The master periodically communicates with each chunk server in HeartBeat messages to give it instructions and collect its state. GFS client code linked into each application implements the file system API and communicates with the master and chunk servers to read or write data on behalf of the application. Clients interact with the master for metadata operations, but all data-bearing communication goes directly to the chunk servers. We do not provide the POSIX API and therefore need not hook into the Linux vnode layer. Neither the client nor the chunk server caches file data. Client caches offer little benefit because most applications stream through huge files or have working sets too large to be cached. Not having them simplifies the client and the overall system by eliminating cache coherence issues. (Clients do cache metadata, however.) Chunk servers need not cache file data because chunks are stored as local files and so Linux's buffer cache already keeps frequently accessed data in memory.

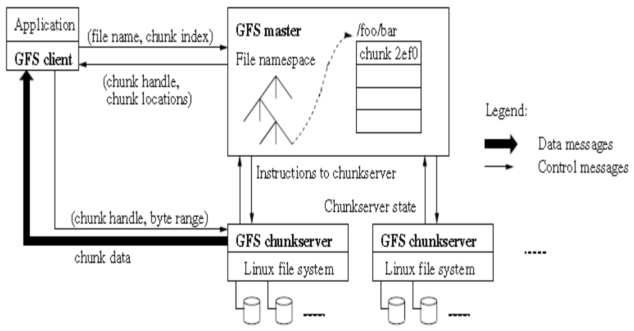

Fig 1: GFS Architecture

HADOOP DISTRIBUTED FILE SYSTEM (HDFS)

HDFS is the file system which is used in Hadoop based distributed file system. The Hadoop is an open-source distributed computing framework and provided by Apache. Many network stations use it to create systems such as Amazon, Facebook. The Hadoop cores are Mapreduce and HDFS. The mapreduce can make the decomposition of tasks and integration of results. The HDFS is a distributed file system and provide the base support for the storage of file in the storage node. The mapreduce provides job trackers and task trackers. Mapreduce is a programming model Google has used successfully in processing big data sets. A map function extracts some intelligence from raw data and a reduce function aggregates according to some guides 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 and which contains nodes and name node. The namenode is a center server and manage the namespace in the file system. The data node manage the data stored in it.

ISSN 2278-3091

International Journal of Advanced Trends in Computer Science and Engineering, Vol. 3 , No.1, Pages : 553– 558 (2014)
Special Issue of ICETETS 2014 - Held on 24-25 February, 2014 in Malla Reddy Institute of Engineering and Technology, Secunderabad– 14, AP, India

ARCHITECTURE OF HDFS

HDFS stores data on the compute nodes, providing very high aggregate bandwidth across the cluster. A HDFS installation consists of single name node as the master node and a number of data nodes as the slave nodes. The name node manages the file system namespace and regulates access to files by clients. The data nodes are distributed, one data node per machine in the cluster, which manage data blocks attached to the machines where they run. The namenode executes the operations on file system namespace and maps data blocks to data nodes. The data nodes are responsible for serving read and write requests from clients and perform block operations upon instructions from namenode [7]. HDFS distributes data chunks and replicas across the server for higher performance, load-balancing and resiliency. With data distributed across all servers, any server may be participating in the reading, writing, or computation of a data-block at any time. HDFS replicates file blocks for fault tolerance. An application can specify the number of replicas of a file at the time it is created, and this number can be changed any time after that.

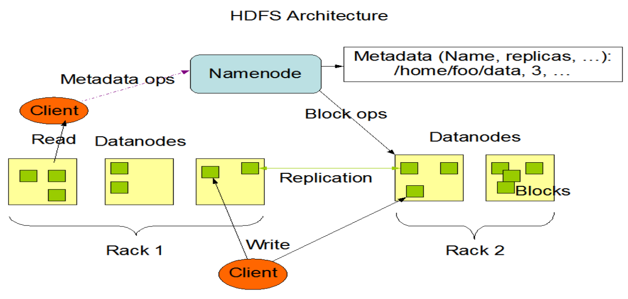

Fig 2: HDFS Architecture

The name node makes all decisions concerning block replication. For a large cluster, it may not be practical to connect all nodes in a flat topology. The common practice is to spread the nodes across multiple racks. Nodes of a rack share a switch, and rack switches are connected by one or more core switches. Communication between two nodes in different racks has to go through multiple switches. In most cases, network bandwidth between nodes in the same rack is greater than network bandwidth between nodes in different racks.

MAPREDUCE

The primary role of Mapreduce is to provide an infrastructure that allows development and execution of large-scale data processing jobs. As such, Mapreduce aims at efficiently exploiting the processing capacity provided by computing clusters while at the same time offering a programming model that simplifies the development of such distributed applications. Moreover and similar to the requirements of GFS, Mapreduce is designed to be resilient to failures such as machine crashes. Google uses mapreduce to process data sets upto multiple terabytes in size for purposes such as indexing web content. To achieve the goals mentioned, Mapreduce has been inspired by the idea of higher order functions, in particular the functions map (also referred to as fold) and reduce. These functions are an integral part of functional programming languages such as Lisp. The primary benefit the functional programming paradigm and these functions in particular promise is to allow the creation of a system that incorporates automatic parallelization of tasks. One of the assumptions made by mapreduce is that all data to be processed can be expressed in the form of key/value pairs and lists of such pairs. Both keys and values are encoded as strings. Based on these assumptions, the key idea of Msapreduce is to implement the application exclusively by writing appropriate map and reduce functions. Provided these functions, the infrastructure not only transparently provides for all necessary communication between cluster nodes, it also automatically distributes and load-balances the processing among the machines. Map is a function written by the user that takes a key/value pair as input and yields a list of key/value pairs as result. A canonical use case for map is thus to digest raw data and generate (potentially very large quantities of) unaggregated intermediate results. Reduce is the second function implemented by the user. It takes a key and a list of values as input and generates a list of values as result. The primary role of reduce is thus to aggregate data.

ISSN 2278-3091

International Journal of Advanced Trends in Computer Science and Engineering, Vol. 3 , No.1, Pages : 553– 558 (2014)
Special Issue of ICETETS 2014 - Held on 24-25 February, 2014 in Malla Reddy Institute of Engineering and Technology, Secunderabad– 14, AP, India

COMPARATIVE ANALYSIS OF GFS AND HDFS

properties	**GFS**	**HDFS**
Design Goals	• The main goal of GFS is to support large files • Built based on the assumption that terabyte data sets will be distributed across thousands of disks attached to commodity compute nodes. • Used for data intensive computing [8]. • Store data reliably, even when failures occur within chunk servers, master, or network partitions. • GFS is designed more for batch processing rather than interactive use by users.	• One of the main goals of HDFS is to support large files. • Built based on the assumption that terabyte data sets will be distributed across thousands of disks attached to commodity compute nodes. • Used for data intensive computing [8]. • Store data reliably, even when failures occur within name nodes, data nodes, or network partitions. • HDFS is designed more for batch processing rather than interactive use by users.
Processes	• Master and chunk server	• Name node and Data node
File Management	• Files are organized hierarchically in directories and identified by path names. • GFS is exclusively for Google only.	• HDFS supports a traditional hierarchical file organization • HDFS also supports third-party file systems such as CloudStore and Amazon Simple Storage Service [9].
Scalability	• Cluster based architecture • The file system consists of hundreds or even thousands of storage machines built from inexpensive commodity parts. • The largest cluster have over 1000 storage nodes, over 300 TB of disk storage, and are heavily accessed by hundreds of clients on distinct machines on a continuous basis.	• Cluster based architecture • Hadoop currently runs on clusters with thousands of nodes. • E.g. Face book has 2 major clusters: - A 1100-machine cluster with 8800 cores and about 12PB raw storage. - A 300-machine cluster with 2400 cores and about 3PB raw storage. - Each (commodity) node has 8 cores and 12 TB of storage. • EBay uses 532 nodes cluster (8*532 cores, 5.3PB) • Yahoo uses more than 100,000 CPUs in >40,000 computers running Hadoop - biggest cluster: 4500 nodes(2*4cpu boxes w 4*1TB disk & 16GB RAM)[10] • K.Talattinis et.al concluded in their work that Hadoop is really efficient while running in a fully distributed mode, however in order to achieve optimal results and get advantage of Hadoop scalability, it is necessary to use large clusters of computers[11]
Protection	• Google have their own file system called GFS. With GFS, files are split up and stored in multiple pieces on multiple machines.	• The HDFS implements a permission model for files and directories that shares much of the POSIX model. • File or directory has separate permissions

	• Filenames are random (they do not match content type or owner). There are hundreds of thousands of files on a single disk, and all the data is obfuscated so that it is not human readable. The algorithms uses for obfuscation changes all the time [12].	for the user that is the owner, for other users that are members of the group, and for all other users [13].
Security	• Google has dozens of datacenters for redundancy. These datacenters are in undisclosed locations and most are unmarked for protection. • Access is allowed to authorized employees and vendors only. Some of the protections in place include: 24/7 guard coverage, Electronic key access, Access logs, Closed circuit televisions, Alarms linked to guard stations, Internal and external patrols, Dual utility power feeds and Backup power UPS and generators [12].	• HDFS security is based on the POSIX model of users and groups. • Currently is security is limited to simple file permissions. • The identity of a client process is just whatever the host operating system says it is. • Network authentication protocols like Kerberos for user authentication and encryption of data transfers are yet not supported [14].
Database Files	• Bigtable is the database used by GFS. Bigtable is a proprietary distributed database of Google Inc.	• HBase[15] provides Bigtable (Google) [16]-like capabilities on top of Hadoop Core.
File Serving	• A file in GFS is comprised of fixed sized chunks. The size of chunk is 64MB. Parts of a file can be stored on different nodes in a cluster satisfying the concepts load balancing and storage management.	• HDFS is divided into large blocks for storage and access, typically 64MB in size. Portions of the file can be stored on different cluster nodes, balancing storage resources and demand [17].
Cache Management	• Clients do cache metadata. • Neither the sever nor the client caches the file data. • Chunks are stored as local files in a Linux system. So, Linux buffer cache already keeps frequently accessed data in memory. Therefore chunk servers need not cache file data.	• HDFS uses distributed cache • It is a facility provided by Mapreduce framework to cache application-specific, large, read-only files (text, archives, jars and so on) • Private (belonging to one user) and Public (belonging to all the user of the same node) Distributed Cache Files [18].
Cache Consistency	• Append-once-read-many model is adapted by Google. It avoids the locking mechanism of files for writing in distributed environment is avoided. • Client can append the data to the existing file.	• HDFS's write-once-read-many model that relaxes concurrency control requirements, simplifies data coherency, and enables high throughput access [9]. • Client can only append to existing files (yet not supported)
Communication	• TCP connections are used for communication. Pipelining is used for data transfer over TCP connections.	• RPC based protocol on top of TCP/IP
Replication Strategy	• Chunk replicas are spread across the racks. Master automatically replicates the chunks. • A user can specify the number of replicas to be maintained. • The master re-replicates a chunk replica as soon as the number of available	• Automatic replication system. • Rack based system. By default two copies of each block are stored by different Data Nodes in the same rack and a third copy is stored on a Data Node in a different rack (for greater reliability) [17].

	replicas falls below a user-specified number.	• An application can specify the number of replicas of a file that should be maintained by HDFS [9]. • Replication pipelining in case of write operations.
Available Implementation	• GFS is a proprietary distributed file system developed by Google for its own use.	• Yahoo, Facebook, IBM etc. are based on HDFS.

CONCLUSION

Google File System is a proprietary distributed file system and is exclusive for Google Inc. Mapreduce is the programming frame work used by Google. Hadoop Distributed File System and Mapreduce are the components of Hadoop project owned by Apache. Hadoop Mapreduce is based on the idea of the Google Mapreduce. In this paper the comparison between these two file systems is made by selecting few parameters.

REFERENCES

1. J. Silvestre. “Economies and Diseconomies of Scale”, The New Palgrave: A dictionary of Economics, v.2, pp. 80-84, 1987.
2. Ghemawat S, Gobioff H, Leung S-T (2003) The Google file system, In : Proc of SOSP, October 2003
3. Hadoop Distributed File System, hadoop.apache.org/hdfs
4. D. Borthakur. HDFS Architecture Guide. [Online]. Available: hadoop.apache.org/docs/hdfs/current/hdfs_design.html
5. S.Ghemawat, H.Gibioff, and S. Leung. (Oct.2003). The Google File System. SOSp ’03 ACM. [Online] Available: http://Research.google.com/archive/gfs.html
6. Information Week, Google Revealed: The IT Strategy that Makes It Work, 08/28/2006 (retrieved 02/01/2008)
7. F.Wang, J.Qiu, J.Yang, B.Dong, X.H.Li, and Y.Li, “Hadoop high availability through metadata replication”, in Proc. The first international workshop on Cloud data management, 2009, pp. 37-44
8. R.T.Kouzes, G.A. Anderson, S.T.Elbert, I.Gorton, and D.K.Gracio, “The changing paradigm of data-intensive computing”, IEEE Computer Society, vol. 42, no. 1, pp. 26-34, Jan.2009
9. J.Hanson. (Feb 2011). An Introduction to Hadoop Distributed File System. [Online]. Available: http://www.ibm.com/developerworks/web/library/wa-introhdfs
10. Hadoop Wiki. [Online]. Available: http://wiki.apache.org/hadoop/PoweredBy
11. K.Talattinis, A.Sidiropoulou, K.Chaalkias, and G.Stephanides, “Parallel collection of live data using hadoop”, in Proc. 14th IEEE Panhellenic Conference on Informatics, pp. 66-71, 2010
12. http://info.groveis.com/blog/bid/178992/On-Site-vs-Google-Cloud-Security-How-do-they-comapare
13. M.Satyanarayanan and M.Spasojevic, “AFS and the web: competitors or collaborators?” in Proc. Of the Seventh ACM SIGOOPS European Workshop, pp. 1-6, 1996
14. HDFS user guide. [Online]. Available: http://hadoop.apache.org/common/docs/current/hdfs_user_guide.html
15. HBase Big Table – like structured storage for Hadoop HDFS. [Online]. Available: http://wiki.apache.org/hadoop/Hbase
16. F.Chang, J.Dean, S.Ghemawat, W.C.Hsieh, D.A.W.M.Burrows, T.Chandra, A.Fikes, and R.E.Gruber, “Bigtable: A distributed storage system for structured data”, in Proc. Of the 7th conference on usenix symposium on operating systems design and implementation – vol. 7, 2006, pp. 1-14.
17. J.Shafer, S.Rixner, and A.L.Cox, “the Hadoop distributed file system: balancing portability and performance”, IEEE International Symposium on Performance Analysis of Systems & Software (ISP ASS), pp. 122-133, 2010
18. MapReduce tutorial. [Online]. Available: http://hadoop.apache.org/common/docs/r0.20.2/mapred_tutorial.html