www.ijecs.in
International Journal Of Engineering And Computer Science ISSN:2319-7242
Volume 3 Issue 9 September, 2014 Page No. 8321-8325

Mapreduce & Comparison of HDFS And GFS

Ameya Daphalapurkar[1], Manali Shimpi[2], Priyal Newalkar[3]

[1]RamraoAdik Institute of Technology, Department of Computer Engineering,
Nerul, Navi Mumbai, Maharshtra, India – 400706.
0411ameya@gmail.com

[2]RamraoAdik Institute of Technology, Department of Computer Engineering,
Nerul, Navi Mumbai, Maharshtra, India – 400706.
mana93cs@gmail.com

[3]RamraoAdik Institute of Technology, Department of Computer Engineering,
Nerul, Navi Mumbai, Maharshtra, India – 400706.
priyaln21@gmail.com

Abstract: Distributed file systems are client based applications in which the central server stores the files that can be accessed via clients with proper authorization rights. Similar to an operating system, the distributed file systems manage the overall system with naming conventions and mapping schemes. Google file system (GFS) was the proprietary system developed by Google for its own use, which included deployment of commodity hardware to retain the enormous generation of data. Hadoop Distributed File System (HDFS), an open source community project, was majorly developed by Yahoo! was designed to store large amounts of data sets reliably along with providing high sets of bandwidths for streaming data on client applications. For the processing of the data stored in HDFS, Hadoop provides the users with a programming model called MapReduce. This model allows the users to reliably distribute a large problem into smaller sub-problems onto several nodes in number of clusters without facing any problems. In this paper, we describe the GFS and HDFS and compare and contrast the two systems based on a list of attributes as also this paper provides the basic functionality of MapReduce framework.

Keywords: HDFS, GFS, MapReduce, distributed file system

1.Introduction

The digital data in terms of images, videos, records are increasing at a fast pace. There is a revolutionary change in the rise of both structured and unstructured data. Organizations require the big data [1] to be stored, managed and processed. The big data storage requires scaling to keep up with the ever increasing growth in the amount of data as well as provide low latency for analytics work.The largest big data practitioners are Google, Facebook, Apple etc. which use hyper scale storage environment [2].

Google was first to face the issues of big data storage. Google came up with Google File System (GFS) as the solution to the problem of storage of big data. It is a scalable distributed file system for large distributed data-intensive applications [3].GFS was designed with many goals common to those of many distributed file systems [3]. Many assumptions such as hardware failures, workloads, high throughput and low latency guided the design of GFS.

The well-known Apache Hadoop project [4] also includes a similar module of distributed file system called as Hadoop Distributed File System (HDFS) to store data on commodity machines. The Apache Hadoop's components – MapReduce and HDFS were originally derived from Google's MapReduceand Google File System (GFS) papers.

2. Comparative Analysis

2.1 Architecture

HDFS [5] and GFS [3] have number of similarities when architecture is taken into consideration. Both the systems involve a master node that controls the overall exaction of the system and the communication between the nodes take place through heartbeats which are messages for instructions and to understand the datanode or chunk's state.

Google file system's[10] cluster includes a single master and a number of chunkservers, which are typically commodity Linux machines which run the user level

server processes. Architecture involves a division of files in fixed size chunks, each of which are identified by a chunkhandle. The master maintains the metadata, mapping of the chunkservers, chunk state information etc.

Hadoop Distributed File System is a cross-platform, scalable file system written in Java. HDFS has a single NameNode and a number of datanodes which serve the network using the protocols specific to HDFS.[wiki] Files in HDFS are split into many blocks which are stored on DataNodes. The entire file system namespace is managed by NameNode.

The difference between implementation of the two file systems is delineated in a nutshell in Table 1.

Table 1: Implementation

	Hadoop Distributed File System	*Google File System*
Platform	Cross-platform	Linux
Written In	Java	C , C++
License	Apache 2.0	Proprietary
Developer(s)	Primary: Yahoo! And also the open source community	Google

The difference and similarity with respect to the architectures of the two systems is explained in table 2.

Table 2: Architecture

	Hadoop Distributed File System	*Google File System*
Node Division	NameNodes & DataNodes	MasterNodes & ChunkServers
Architectural paradigm	Complete view of the file system is available for the NameNode.	Master stores files and locations and makes global policy decisions regarding storage of chunks on servers or racks.
Hardware Utilization	Commodity hardware or servers	
Inter-Node Communication	NameNode and MasterNode both use periodic heartbeats to convey commands to ChunkServers or DataNodes	
ChunkServers or DataNodes	Server process at user level store chunks in local file system as files.	

2.2 File System State

In GFS[3], the client application translates the file name it specifies into chunk index within a file using the fixed size chunks. The request is then forwarded to the master who contains the file name and chunk index and the request is replied with a chunk handle and location.

In HDFS, the NameNode preserves the mapping of the files and the DataNode is consistently flushed with file index state and it results in the modifications of logs and records.

The Table 3 casts differences between the indexing of the files and chunks and on the methods for verifying data integrity.

Table 3: File System State

	Hadoop Distributed File System	*Google File System*
File Index State	File index state and mapping of files to chunks kept in memory at NameNode and periodically flushed to disk; modification log records changes in between flushing.	The chunk size is used by the client to translate the file name into a chunk index which is later requested to the master along with a file name.
Chunk State and Location	Chunk location information is consistently maintained by the NameNode.	Chunk location handle and location of replicas is replied by the master to the requesting client, which the client caches by using the index and filename as a key.
Data Integrity	Data that is written by a client is sent to a pipeline of DataNodes and the checksum is verified by the last DataNode in the pipeline.	ChunkServers use checksums to detect corruption of the stored data. Comparison of the replicas is another alternative.

2.3 File System Operations

In GFS[9], the default size of the chunks is 64mb. GFS provides with operations such as record appends and delete operations and has a unique garbage collection process.

In HDFS, default block size is 128mb and it supports only append operations. The deleted files are renamed and moved into folders from which they are later subject to a lazy garbage collection process[10].

Table 4: Operations

	Hadoop Distributed File System	*Google File System*
Write Operations	Supports only append.	Along with append operation, even random offset writes and record appends are performed
Write Consistency Guarantees	Consists of a single-writer model which assures that the files are always defined and consistent.	• Undefined region are created by successful concurrent writes. • Defined regions are a result of successful concurrent appends.
Deletion	Deleted files are renamed into a particular folder and are then removed via garbage collection process.	Unique garbage collection process. The resources of deleted files are not reclaimed immediately and are renamed in the hidden namespace which are further deleted if they are found existing for 3 days of regular scan.
Snapshots	Up to 65,536 snapshots allowed for each directory in HDFS 2.	Individual files and directories can be snapshotted in GFS.
Default Size	128 MB default but it can be altered by the user.	64 MB default but it can be altered by the user.

3. MapReduce Functionality

Google first developed the programming model and software framework called MapReduce to process large sets of data. It was designed to process the records from the user in parallel via Mappers and then merge the outputs of these Mappers with the help of Reducers. MapReduce technique follows a divide-and-conquer strategy while executing the data-centric applications. Implementation of applications using MapReduce takes place in two phases viz. map and reduce phase[7].

Based on the notion of parallel programming the MapReduce functionality is implemented with the help of centralized master/slave architecture. The architecture employs a single master node (job tracker) and several slave nodes (task tracker). The job tracker is in charge of scheduling the jobs' tasks on the slaves, monitoring the performance of the slaves and re-executing the tasks experiencing any kind of failure and acts like an interface between the user and Hadoop framework. Whereas, the task trackers are responsible for executing the tasks as guided by the job tracker (master) and manage the data flow between the two phases[6].

3.1 Core functionality Of Map Phase

In this phase, user submits the large problem input to the job tracker. Job tracker assigns the problem input to the various task trackers requesting to the job tracker (pull mechanism) , by slicing the large input into smaller chunks of data (<key, value> pairs) using the map function that can be processed in isolation. Task trackers receive the tasks based on the number of free slots indicated to the master via the heartbeat protocol.[6] Task trackers on receiving their respective tasks read the input stored in HDFS. These workers (slaves) execute these smaller sub-problems and then back to the master node. Mapper maps the input key-value pairs to generate the intermediate key-value pairs. The outputs of the maps are then sorted by the framework and stored on a local storage for easy access to the reducers [8].

Combiner functions are sometimes used on the output of map phase to improve the efficiency by saving the data-transfer bandwidth. Local aggregation is performed by these functions by grouping the intermediate values corresponding to a particular output value[7].

3.2 Core functionality of Reduce phase

The reducer program reads all the intermediate results from the local storage having the same key values and invokes the reduce function to generate a smaller single solution. The reduce function is supplied by the user.[8] The output writer collects the results of the reduce program which are then written back and stored in temp files HDFS. On completion of all the reduce tasks this temp file is automatically renamed to its final destination file name thereby terminating the operation of parallel processing.

The task trackers report their status to the job trackers after specific time-intervals for the job trackers to monitor the progress of the process currently in execution on several nodes. In case, if the job tracker fails to receive the status information from any of the task tracker for a certain amount of time, the job tracker assumes that the task tracker node has failed and thereby reassigns the task to some other available task tracker. If the task tracker fails in execution during the reduce phase then only the incomplete reduce operation is being re-executed by some other slave node[6].

Following figure explains the basic MapReduce model.

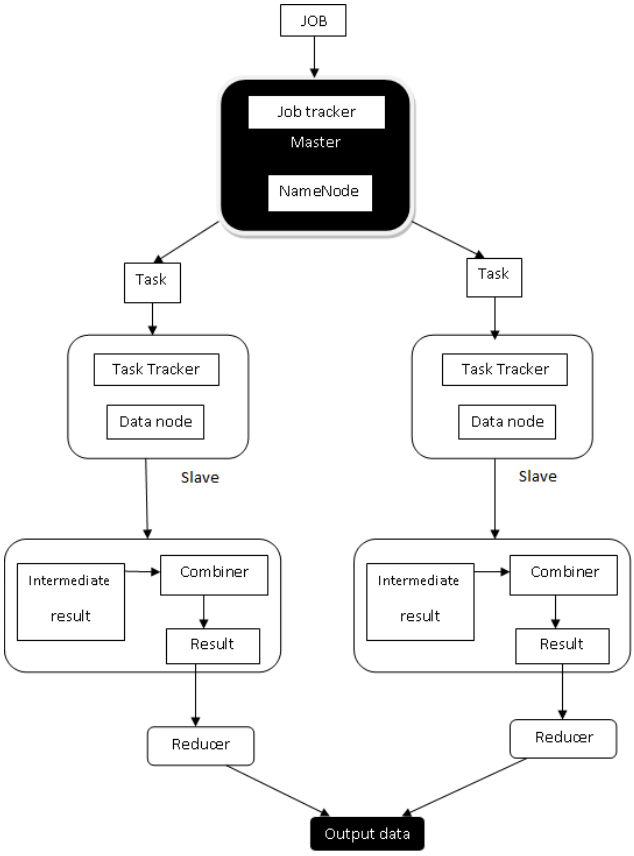

Figure 1: MapReduce model

4. Conclusion

GFS and HDFS are similar in many aspects and are used for storing large amounts of data sets. HDFS being a module of an open source project (Hadoop) it is vastly applicable (Yahoo!, Facebook, IBM etc use HDFS) as compared to the proprietary GFS.

MapReduce provides distributed, scalable and data-intensive computing. It is reliable and fault-tolerant and provides load-balancing as processing of a large task is done in several clusters simultaneously [7].

On the other hand, there are problems that cannot be divided into isolated sub-problems cannot be processed using MapReduce. As Jobs run in isolation in MR, if the processes need to communicate with each other during the processing then it is difficult in MapReduce. Streamed data is difficult to handle using MapReduce.

In this paper the comparison between GFS and HDFS is made on the basis of few parameters.

5. References

[1] http://en.wikipedia.org/wiki/Big_data. [Accessed: Sept. 8, 201

[2] http://en.wikipedia.org/wiki/Hyperscale. [Accessed: Sept. 10, 2014

[3] S. Ghemawat, H. Gobioff, S.-T. Leung, The Google File System, ACM SOSP10/2003.

[4] http://hadoop.apache.org/. [Accessed: Sept. 12, 2014]

[5] K. Shvachko, H. Kuang, S. Radia and R. Chansler, "The Hadoop Distributed File System", IEEE 26th Symposium on Mass Storage Systems and Technologies (MSST), **(2010)** May 3-7: Incline Village, USA.

[6] Heger, D., "Hadoop Design, Architecture & MapReduce Performance", CMG Journal of Computer Resource Management,December 2012.

[7] http://www.opensourceforu.com/2011/03/mapreduce-more-power-less-code-hadoop/[Accessed: Sept. 8, 2014]

[8] Casey McTaggart, "Object-oriented framework presentation",CSCI 5448.

[9] Chen, Y.Ganapathi, A.R.Griffith and Katz R., "The Case for Evaluating MapReduce Performance Using Workload Suite." {\textit{Modeling, Analysis and Simulation of Computer and Telecommunication System, Singapore, 25-27 July 2011. Page(s)::390 - 399}}

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

Author Profile

Ameya Daphalapurkar , a student of B.E. computer engineering at Ramrao Adik Institute of Technology, navi Mumbai,Maharashtra,India.

Manali Shimpi , a student of B.E. computer engineering at Ramrao Adik Institute of Technology, navi Mumbai,Maharashtra,India.

Priyal Newalkar , a student of B.E. computer engineering at Ramrao Adik Institute of Technology, navi Mumbai,Maharashtra,India.