Big Data Management Processing with Hadoop MapReduce and Spark Technology: A Comparison

Ankush Verma
Pacific University
Udaipur, Rajashthan
ankush.verma08@rediffmail.com

Ashik Hussain Mansuri
Pacific University
Udaipur, Rajasthan
aashiqmansuri@gmail.com

Dr. Neelesh Jain
Sagar Insitiute of Research & Technology, Bhopal
neeleshcmc@gmail.com

***Abstract* – Hadoop MapReduce is processed for analysis large volume of data through multiple nodes in parallel. However MapReduce has two function Map and Reduce, large data is stored through HDFS. Lack of facility involve in MapReduce so Spark is designed to run for real time stream data and for fast queries. Spark jobs perform work on Resilient Distributed Datasets and directed acyclic graph execution engine. In this paper, we extend Hadoop MapReduce working and Spark architecture with supporting kind of operation to perform. We also show the differences between Hadoop MapReduce and Spark through Map and Reduce phase individually.**

Keywords – Big data Analysis; Hadoop MapReduce; HDFS; Spark; Resilient Distributed Datasets; Directed Acyclic Graph.

I. INTRODUCTION

Large amount of data generated by number of users worldwide. This data is on various forms mostly the large data is unstructured .The huge data is keep to store and analyze every day. In addition new technologies have been emerged to exploding volumes of complex data, including web traffic, social media content, machine generated data, sensor data and system data. Success of such Big Data depends on its analysis. Since its early beginnings from 8 years ago, the MapReduce Hadoop implementation has become for storing, managing and processing massively large data volumes. Hadoop MapReduce has come up as a highly effective and efficient tool for analysis Big Data. MapReduce is also ideal for scanning historical data and performing analytics.

But MapReduce is not fast in response and for real time analysis that type of structure is done by Spark which is widely used in Big Data. Developed in 2009 and open sourced in 2010, Apache Spark, unlike MapReduce, is all about performing sophisticated analytics at lightning fast speed. Today, as organizations facing the growth need for real-time data analysis to achieve advantage, a new open source Hadoop data processing engine, Apache Spark, has entered in the field.

Numbers of analytical techniques are available in the market for analysis of complex structured, semi structured and unstructured data in real time. Whereas MapReduce adopts a different model and supports to control running task on each node. In this conventional Hadoop environment, data storage and computation both reside on the same physical nodes within the cluster and is designed to run batch jobs that address every file in the system.

II. HADOOP MAPREDUCE

The Hadoop is open source java based framework is designed to provide shared storage and analysis infrastructure. Its cluster is able to store data and perform parallel computation across a large computer cluster having a single master and multiple nodes. The Hadoop MapReduce (MR) act as a master - slave architecture where one master node manages a number of slave nodes. Hadoop is the basically the combination of HDFS and MR.

Hadoop = HDFS + MR

A. MapReduce

The MapReduce framework has two basic operations on the data i.e. Mapper and Reducer function [1].

1. Mapper

It is applied to all input key-value pairs, which generate a number of intermediate key-value pairs. Mapper next sorts the set of key-value pairs by their keys. Combiner can emit any number of key-value pairs, but the keys and values must be of the same type as the mapper output then Partitioners are responsible for dividing up the intermediate key space and assigning intermediate key-value pairs to reducers [2] [6].

2. Reducer

The Reducer's first job is to gather all of its input from the various Mapper output partitions. Sort is functioning here the shuffle and sort occurs simultaneously from different group. After Shuffle and Sort reduce work the output of the reduce task is return on file system.

B.HDFS

It is not possible for storing large amount of data on a single node, therefore Hadoop use a new file system called HDFS which split data into many smaller parts and

978-1-5090-0669-4/16/$31.00 © 2016 IEEE

distribute each part across multiple nodes. The HDFS is designed to store very large data sets and to stream data sets to user applications. In a large cluster, thousands of servers both host directly attached storage and execute user application tasks [3]. HDFS have two types of nodes in HDFS: the DataNodes (Master) and the NameNode (Worker). They support operations to read, write and delete files, and operations to create and delete directories [4]. The Name Node is contacted to request access permission. If granted, the Name Node will translate the HDFS filename into a list of the HDFS block IDs comprising that file and a list of DataNodes that store each block, and return the lists to the client.

C. Hadoop MapReduce Working

The architecture utilizes a single master server (JobTracker) and several slave servers (TaskTracker's). The JobTracker represents a centralized program that keeps track of the slave nodes, and provides an interface infrastructure for job. When the actual data is normally stored, the JobTracker reflects the interaction point among the users and the framework. Users submit MR jobs to the JobTracker, which inserts the jobs into the pending jobs queue and executes them on a FIFO basis. JobTracker manages then and reduces task assignments with the TaskTracker's. The TaskTracker's execute the jobs based on the instructions from the Job Tracker and handle the data movement between the maps and reduce phases.

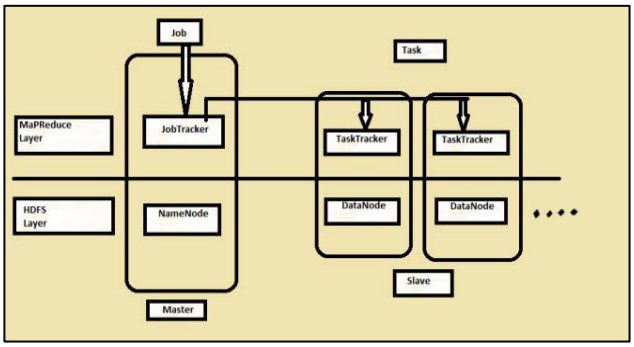

Fig 1: Hadoop Working

The JobTracker monitors the progress of running MR jobs and is responsible for coordinating the execution of the mappers and reducers. A Hadoop MR job is divided up into a number of map tasks and reduce tasks. Each assigned TaskTracker will fork a Map Task to execute the map processing cycle. After the Map Task finished executing all input records, data pairs and index is merge into a single unit [5]. As on the Map Task is complete the JobTracker start initiate reduce task. The JobTracker give instruction to TaskTracker and TaskTracker download the completed files from the map task nodes, and basically concatenate the files into a single entity.

III. SPARK

Spark is designed to run on top of Hadoop and it is an alternative to the traditional batch map and reduce model that can be used for real time stream data processing and fast queries that finish work within a seconds. In addition to Map and Reduce functions spark take supports of SQL queries, streaming data, machine learning and graph data processing for big data analysis. The spark platform is implemented in Scala and is hence run within the Java Virtual Machine (JVM). In addition to a Scala API, interfaces in Java and Python are available. Spark provides two options for running applications. Firstly, interpreter in the Scala language distribution allows users to run queries on large data sets through Spark engine. Secondly applications can be written as Scala programs called driver programs and can be submitted to the cluster's master node after compilation [7].

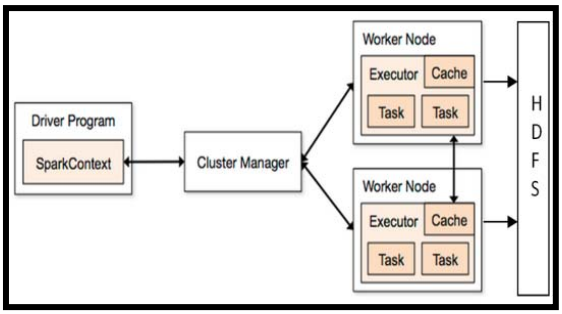

Fig 2: Spark Architecture

Apache spark consists of a driver program (SparkContext), workers also called executors, cluster manager, and the HDFS [10]. Driver program is the main program of spark. Spark applications run as independent sets of processes on a cluster, coordinated by the SparkContext object called the driver program. Whereas each application gets its own processes and run tasks in multiple threads and must be network addressable from worker nodes. Once connected, Spark acquires executors on nodes in the cluster, which are worker processes that run computations and store data for your application. Next, it sends your application code (defined by JAR or Python files passed to SparkContext) to the executors. Finally, SparkContext sends tasks for the executors to run [10] [12].

Apache Spark is based on two key concepts; Resilient Distributed Datasets (RDD) and directed acyclic graph (DAG) execution engine. With regard to datasets, Spark supports two types of RDDs: parallelized collections are based on Scala collections and Hadoop datasets that are created from the files which are stored on HDFS.

A.Resilient Distributed Dataset

Spark jobs perform work on Resilient Distributed Datasets (RDD), an abstraction for a collection of elements that can be operated on in parallel. When Spark is running on a Hadoop cluster, RDDs are created from files in the distributed file system in any format such as text and sequence files or anything supported by a Hadoop forma [9]. A resilient distributed dataset (RDD) is a read-only collection of objects partitioned across a set of machines that can be rebuilt if a partition is lost. The elements of an RDD need not exist in physical storage. RDDs support two kinds of operations: transformations and actions [11].

1. Transformation

When transformations are applied on RDD's, they return a new RDD and not a single RDD. Nothing is evaluated when call a Transformation function it just takes an RDD and return a new RDD. They are executed only when an action runs on it. Some of the Transformation functions are map, filter, flatMap, groupByKey, reduceByKey, aggregateByKey, pipe and coalesce [8].

2. Action

Action operations are applied on RDD's it evaluates and returns a new value. All the data processing queries are computed at that time and the result value is returned. Some of the Action operations are reduce, collect, count, first, take, countByKey, and for each [8].

B. Directed Acyclic Graph

Spark consists of an advanced Directed Acyclic Graph (DAG) engine which supports cyclic data flow. Each Spark job creates a DAG of task stages to be performed on the cluster. DAG created with in two stages - Map and Reduce. This allows simple jobs to complete after just one stage, and more complex tasks to complete in a single run of many stages, rather than splitting it into multiple jobs. Thus, jobs complete faster than they would in MapReduce. [8][9].

IV. DIFFERENCE BETWEEN HADOOP MAPREDUCE AND SPARK

Table 1: Difference between MapReduce and Spark

Hadoop MapReduce	***Spark***
Hadoop MapReduce stores data on disk	Spark stores data in-memory i.e. first whole data is store into memory and then process it
disk based memory computing i.e. partial use of memory	RAM memory based computing i.e. partial use of disk
It is for batch only	support interactive query
MapReduce uses replication for fault tolerance	Resilient Distributed Datasets (RDD) for fault tolerance
It is used for generating report help to find answer from historical queries	It perform streaming, batch processing and machine learning
It is slower than spark	It is 100 time faster than MapReduce in execution
It is highly painful when it comes to processing and analyzing data in real time	Spark can be used to modify the data in real time
Hadoop handles in low cost	Cost-wise Spark requires a more cost in
MapReduce is faster in batch processing	If the data size is greater than memory it will be far slower in the batch processing
It merge and partition shuffle spill files into one big file	SPARK doesn't merge and partition shuffle files
MapReduce is inefficient for applications that repeatedly reuse a same set of data	Keep working sets in memory for efficient reuse
Writing Hadoop MapReduce code is complex and lengthy	Writing Spark code is always compact

A. Map and Reduce phase in MapReduce

The Map phase of Hadoop MapReduce some steps are shown in explained below [12]:

- Input the file
- The Map phase in the form of Key and Value pairs.
- The output of pair is stored in a buffer memory.
- When the buffer fills 80% then it data is spilled onto disk.
- All the spill files are combined into a single big file which is partitioned and sorted depending upon the reducers.

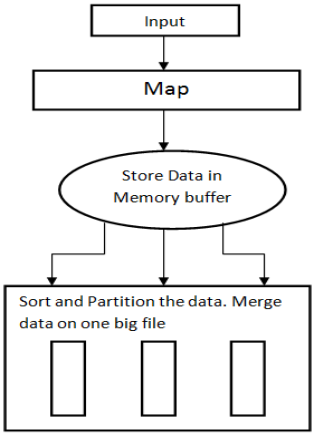

Fig 3: Map Phase in MapReduce

Reduce function for Hadoop MapReduce [12]:

- The data from the map phase is pulled by the reducers and loaded into the memory.
- If buffer reaches 70% it is spilled onto disk.
- The data spilled to the disk then merged and sort into larger files
- The reduce function is initiated and reduce the file.

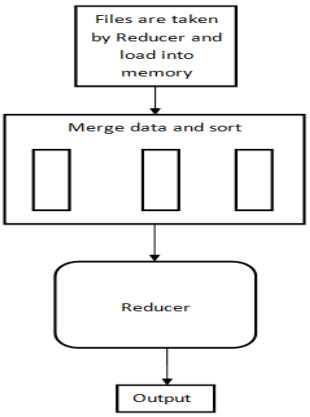

Fig 4: Reduce Phase in MapReduce

B. Map and Reduce phase in Spark

The steps for the Map phase in Spark are explained below [13]:

- The output of map phase is written to Operating System disk Buffer Cache.
- Operating System decides whether the data will stay in buffer or will be spilled onto disk.
- Spark does not merge them into a single, partitioned one.

- Each Map task outputs as many spill files as number of reducers.

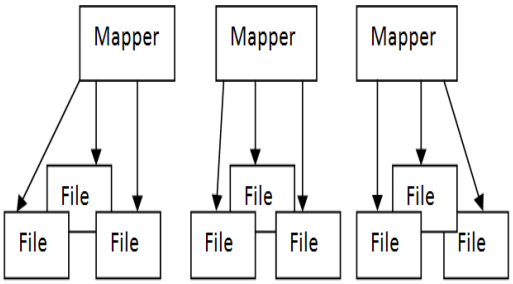

Fig 5: Map Phase in Spark

Step for Reduce phase in Spark [14]:

- After map phase Map push the data to all shuffle function to fit in memory of the corresponding reduce task.
- These files are written to reducer's memory and reduce functionality is invoked.

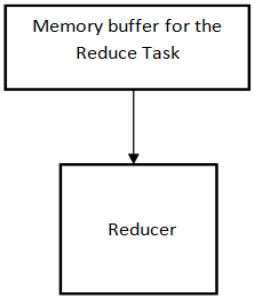

Fig 6: Reduce Phase in Spark

V. CONCLUSION

As large scale of data from different resources to analysis is inefficient execution in Hadoop MapReduce was observed. Our proposed Benefit to manage tasks in a benefit-aware manner and expected to improve the efficiency. Finally various algorithms which take into consideration into the real-time network are proposed to improve the performance caused by network layer. They Considering the above mentioned advantages and limitations, Apache Spark was able to analyze data in few seconds. Alternative processing frameworks there's still a lot of tooling for MapReduce. Comparison for this analysis shows that Spark is a very strong using in-memory processing and fast. Observing that Spark's ability to perform batch processing, streaming, and machine learning. It is a framework for a processing large number data.

REFERENCES

[1] Jeffrey Dean and Sanjay Ghemawa, "MapReduce: Simplified Data Processing on Large Clusters", Communications of the ACM, Vol 51, Jan 2008.
[2] Jimmy Lin and Chris Dyer, "Data-Intensive Text Processing with Map Reduce", P. 18-38, April 2010.
[3] Online article : http://developer.yahoo.com/blogs/hadoop/posts/2008/07/apache_hadoop_wins_terabyte_sort_benchmark
[4] Dr. Siddaraju, C. L. Sowmya, K. Rashmi and M. Rahul, "Efficient Analysis of Big Data Using Map Reduce Framework", International Journal of Recent Development in Engineering and Technology, Vol. 2, June 2014.
[5] Dominique A. Heger, "Hadoop Design, Architecture & MapReduce Performance".
[6] Donald Miner and Adam Shook, "MapReduce Design Patterns", P 2-6.
[7] Stefan Nuesch, "Real-Time Anomaly Detection in Water Distribution Networks uses Spark streaming", November 2014.
[8] Amol Bansod, "Efficient Big Data Analysis with Apache Spark in HDFS", IJEAT, Vol. 4, August 2015.
[9] Online article : https://www.mapr.com/products/product-overview/apache-spark
[10] Online article : http://spark.apache.org/docs/latest/cluster-overview.html
[11] Istvan Szegedi,Apache Spark: a fast big data analytics engine Online article : https://dzone.com/articles/apache-spark-fast-big-data
[12] Tom White, "Hadoop the definitive guide", p 167-180.
[13] Aaron Davidson, Andrew Or, "Optimizing Shuffle Performance in Spark".
[14] Nirali Rana, Shyam Deshmukh, "Shuffle Performance in Apache Spark", IJERT, Vol. 4, February, 2015.

.