RESEARCH ARTICLE OPEN ACCESS

A Review Study of Apache Spark in Big Data Processing

V Srinivas Jonnalagadda [1], P Srikanth [2], Krishnamachari Thumati [3]
Sri Hari Nallamala [4]
Assistant Professors [1], [2], [3]
Department of Computer Science and Engineering
DVR & Dr.HS MIC College of Technology
Kanchikacherla, Krishna Dist
Assistant Professor [4]
Department of Computer Science and Engineering
DVR & Dr. HS MIC College of Technology, Kanchikacherla & Research Scholar,
Dept. of CSE, K L University, Guntur
Andhra Pradesh - India

ABSTRACT
Why Spark becomes a hot topic in Big Data analytics? Is really Apache Spark going to replace Hadoop? If we involved seriously into Big Data analytics, then, should we really care about Spark? Apache Spark is a lightning-fast cluster computing designed for fast computation. It was built on top of Hadoop MapReduce and it extends the MapReduce model to efficiently use more types of computations which includes Interactive Queries and Stream Processing.
Keywords :- *Apache Spark, Apache Hadoop, Big Data, MapReduce, RDD, Open Source.*

I. INTRODUCTION

Industries are using Hadoop extensively to analyze their data sets. The reason is that Hadoop[1] framework is based on a simple programming model (MapReduce) and it enables a computing solution that is scalable, flexible, fault-tolerant and cost effective. Here, the main concern is to maintain speed in processing large datasets in terms of waiting time between queries and waiting time to run the program. Spark is a framework for performing general data analytics on distributed computing cluster like Hadoop. It provides in memory computations for increase speed and data process over mapreduce. It runs on top of existing hadoop cluster and access hadoop data store (HDFS), can also process structured data in Hive and Streaming data from HDFS, Flume, Kafka, Twitter.

In this paper, we will see the brief descriptions of Spark, its features and working with Spark using Hadoop.

II. EVOLUTION OF APACHE SPARK

Spark[4] was introduced by Apache Software Foundation for speeding up the Hadoop computational computing software process. As against a common belief, **Spark is not a modified version of Hadoop** and is not, really, dependent on Hadoop because it has its own cluster management. Hadoop is just one of the ways to implement Spark. Spark uses Hadoop in two ways – one is **storage** and second is **processing**. Since Spark has its own cluster management computation, it uses Hadoop for storage purpose only. The main feature of Spark is its **in-memory cluster computing** that increases the processing speed of an application. Spark is designed to cover a wide range of workloads such as batch applications, iterative algorithms, interactive queries and streaming. Apart from supporting all these workload in a respective system, it reduces the management burden of maintaining separate tools.

Spark is one of Hadoop's sub project developed in 2009 in UC Berkeley's AMPLab by Matei Zaharia. It was Open Sourced in 2010 under a BSD license. It was donated to Apache software foundation in 2013, and now Apache Spark has become a top level Apache project from Feb-2014.

III. IS APACHE SPARK GOING TO REPLACE HADOOP?

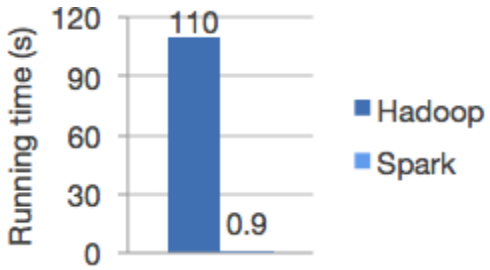
Hadoop is parallel data processing framework that has traditionally been used to run map/reduce jobs. These are long running jobs that take minutes or hours to complete. Spark has designed to run on top of

Hadoop and it is an alternative to the traditional batch map/reduce model that can be used for real-time stream data processing and fast interactive queries that finish within seconds. So, Hadoop supports both traditional mapreduce and Spark. We should look at Hadoop as a general purpose Framework that supports multiple models and we should look at Spark as an alternative to Hadoop MapReduce[10] rather than a replacement to Hadoop.

IV. FEATURES OF APACHE SPARK

Apache Spark[6] has following features.

- **Speed** − Spark helps to run an application in Hadoop cluster, up to 100 times faster in memory, and 10 times faster when running on disk. This is possible by reducing number of read/write operations to disk. It stores the intermediate processing data in memory. It uses the concept of a Resilient Distributed Dataset (RDD), which allows it to transparently store data on memory and persist it to disc only it's needed. This helps to reduce most of the disc read and write – the main time consuming factors – of data processing.



- **Ease of Use** – Spark[5] lets you quickly write applications in Java, Scala, or Python. This helps developers to create and run their applications on their familiar programming languages and easy to build parallel apps. It comes with a built-in set of over 80 high-level operators. We can use it interactively to query data within the shell too.

  Word count in Spark's Python API

```
datafile = spark.textFile("hdfs://...")
datafile.flatMap(lambda line: line.split())
        .map(lambda word: (word, 1))
        .reduceByKey(lambda a, b: a+b)
```

- **Combines SQL, streaming, and complex analytics** – In addition to simple "map" and "reduce" operations, Spark supports SQL queries, streaming data, and complex analytics such as machine learning and graph algorithms out-of-the-box. Not only that, users can combine all these capabilities seamlessly in a single workflow.

- **Supports multiple languages** − Spark provides built-in APIs in Java, Scala, or Python. Therefore, you can write applications in different languages. Spark comes up with 80 high-level operators for interactive querying.

- **Advanced Analytics** − Spark not only supports 'Map' and 'reduce'. It also supports SQL queries, Streaming data, Machine learning (ML), and Graph algorithms.

- **Runs Everywhere** − Spark runs on Hadoop, Mesos, standalone, or in the cloud. It can access diverse data sources including HDFS, Cassandra, HBase etc.

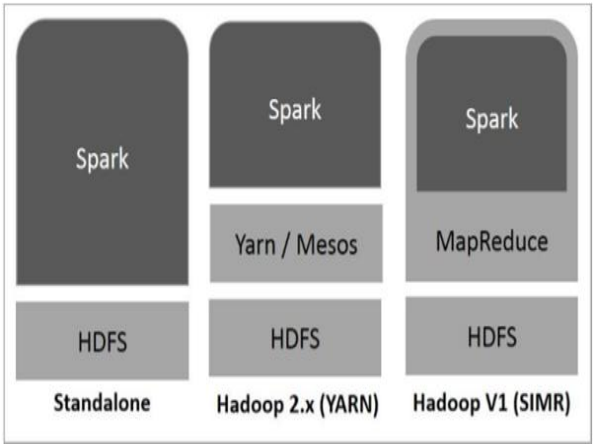
V. SPARK'S MAJOR USE CASES OVER HADOOP

- Iterative Algorithms in Machine Learning.
- Interactive Data Mining and Data Processing.
- Spark is a fully Apache Hive-compatible data warehousing system that can run 100x faster than Hive.
- Stream processing: Log processing and Fraud detection in live streams for alerts, aggregates and analysis.
- Sensor data processing: Where data is fetched and joined from multiple sources, in-memory dataset really helpful as they are easy and fast to process.
  Note: Spark (beta version) is still working out bugs as it matures.

VI. SPARK IN BIG DATA PROCESSING – COMPONENTS, OPERATIONS & ISSUES

A. Spark Built on Hadoop

The following diagram shows three ways of how Spark can be built with Hadoop[2] components.



There are three ways of Spark deployment[7] as explained below.

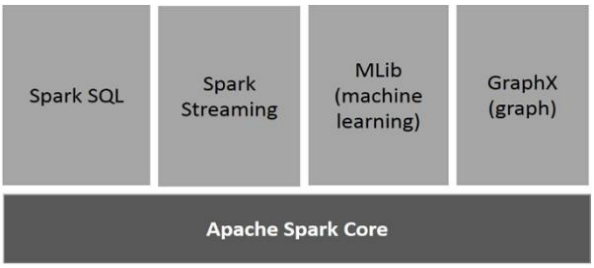
- **Standalone** – Spark Standalone deployment means Spark occupies the place on top of HDFS[3] (Hadoop Distributed File System) and space is allocated for HDFS, explicitly. Here, Spark and MapReduce will run side by side to cover all spark jobs on cluster.

- **Hadoop Yarn** – Hadoop Yarn[9] deployment means, simply, spark runs on Yarn without any pre-installation or root access required. It helps to integrate Spark into Hadoop ecosystem or Hadoop stack. It allows other components to run on top of stack.

- **Spark in MapReduce (SIMR)** – Spark in MapReduce is used to launch spark job in addition to standalone deployment. With SIMR, user can start Spark and uses its shell without any administrative access.

B. Components of Spark

The following illustration depicts the different components of Spark.



- **Apache Spark Core**
  Spark Core is the underlying general execution engine for spark platform that all other functionality is built upon. It provides In-Memory computing and referencing datasets in external storage systems.

- **Spark SQL**
  Spark SQL is a component on top of Spark Core that introduces a new data abstraction called SchemaRDD, which provides support for structured and semi-structured data.

- **Spark Streaming**
  Spark Streaming leverages Spark Core's fast scheduling capability to perform streaming analytics. It ingests data in mini-batches and performs RDD (Resilient Distributed Datasets) transformations on those mini-batches of data.

- **MLlib (Machine Learning Library)**
  MLlib is a distributed machine learning framework above Spark because of the distributed memory-based Spark architecture. It is, according to benchmarks, done by the MLlib developers against the Alternating Least Squares (ALS) implementations. Spark MLlib is nine times as fast as the Hadoop disk-based version of **Apache Mahout** (before Mahout gained a Spark interface).

- **GraphX**
  GraphX is a distributed graph-processing framework on top of Spark. It provides an API for expressing graph computation that can model the user-defined graphs by using Pregel abstraction API. It also provides an optimized runtime for this abstraction.

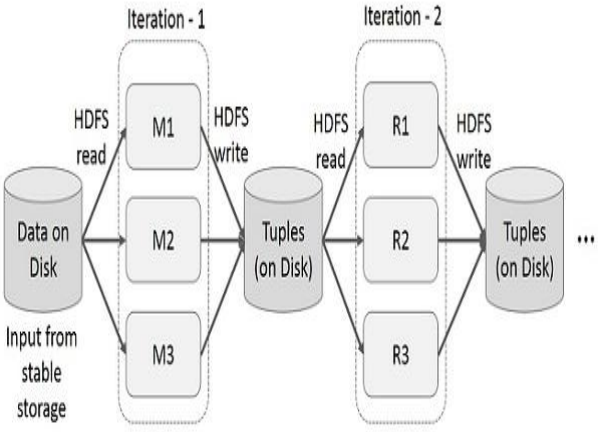
C. Apache Spark Operations – RDD & MapReduce

- **Resilient Distributed Datasets**
  Resilient Distributed Datasets (RDD) is a fundamental data structure of Spark. It is an immutable distributed collection of objects. Each dataset in RDD is divided into logical partitions, which may be computed on different nodes of the cluster. RDDs can contain any type of Python, Java, or Scala objects, including user-defined classes.
  Formally, an RDD is a read-only, partitioned collection of records. RDDs can be created through deterministic operations on either data on stable storage or other RDDs. RDD is a fault-tolerant collection of elements that can be operated on in parallel.
  There are two ways to create RDDs – **parallelizing** an existing collection in your driver program, or **referencing a dataset** in an external storage system, such as a shared file

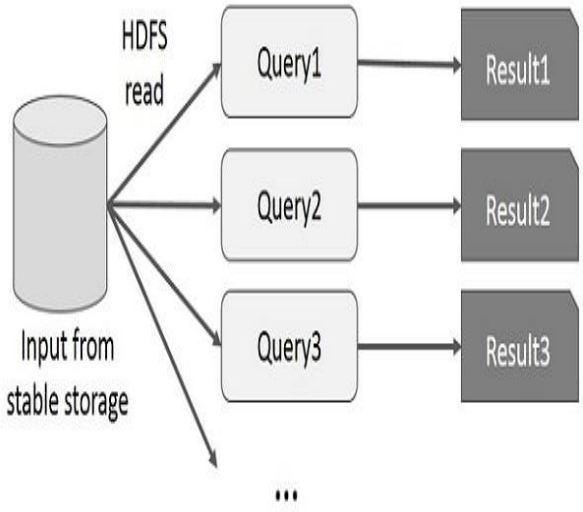
system, HDFS, HBase, or any data source offering a Hadoop Input Format.
Spark makes use of the concept of RDD to achieve faster and efficient MapReduce operations. Let us first discuss how MapReduce operations take place and why they are not so efficient.

- **Data Sharing is Slow in MapReduce**
  MapReduce is widely adopted for processing and generating large datasets with a parallel, distributed algorithm on a cluster. It allows users to write parallel computations, using a set of high-level operators, without having to worry about work distribution and fault tolerance.
  Unfortunately, in most current frameworks, the only way to reuse data between computations (Ex – between two MapReduce jobs) is to write it to an external stable storage system (Ex – HDFS). Although this framework provides numerous abstractions for accessing a cluster's computational resources, users still want more.
  Both **Iterative** and **Interactive** applications require faster data sharing across parallel jobs. Data sharing is slow in MapReduce due to **replication, serialization**, and **disk IO**. Regarding storage system, most of the Hadoop applications, they spend more than 90% of the time doing HDFS read-write operations.

- **Iterative Operations on MapReduce**
  Reuse intermediate results across multiple computations in multi-stage applications. The following illustration explains how the current framework works, while doing the iterative operations on MapReduce. This incurs substantial overheads due to data replication, disk I/O, and serialization, which makes the system slow.



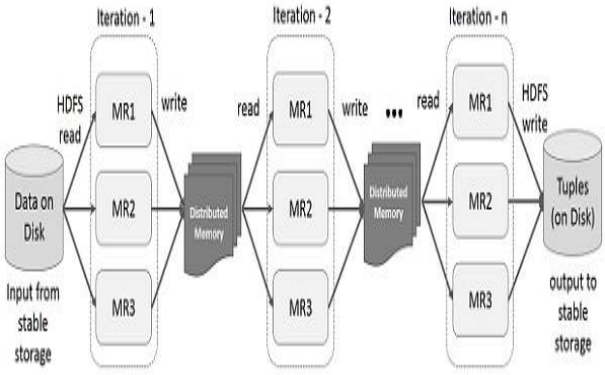
- **Interactive Operations on MapReduce**
  User runs ad-hoc queries on the same subset of data. Each query will do the disk I/O on the stable storage, which can dominates application execution time.
  The following illustration explains how the current framework works while doing the interactive queries on MapReduce.



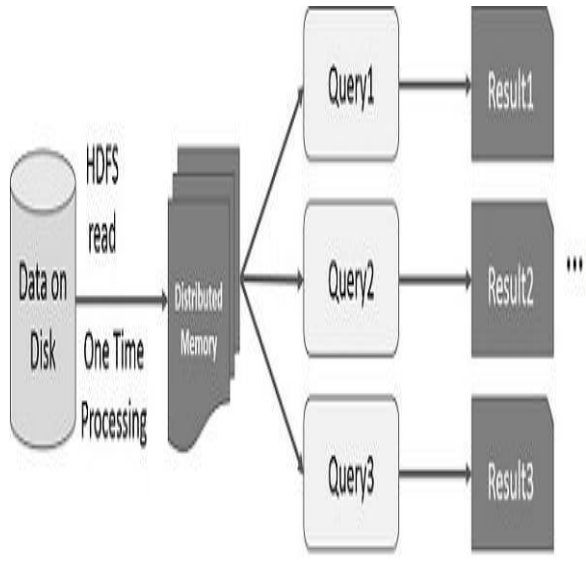
- **Data Sharing using Spark RDD**
  Data sharing is slow in MapReduce due to **replication, serialization**, and **disk IO**. Most of the Hadoop applications, they spend more than 90% of the time doing HDFS read-write operations.
  Recognizing this problem, researchers developed a specialized framework called Apache Spark. The key idea of spark is **R**esilient **D**istributed **D**atasets (RDD); it supports in-memory processing computation. This means, it stores the state of memory as an object across the jobs and the object is sharable between those jobs. Data sharing in memory is 10 to 100 times faster than network and Disk.
  Let us now try to find out how iterative and interactive operations take place in Spark RDD.

- **Iterative Operations on Spark RDD**
  The illustration given below shows the iterative operations on Spark RDD. It will store intermediate results in a distributed memory instead of Stable storage (Disk) and make the system faster.

  Note − If the Distributed memory (RAM) is sufficient to store intermediate results (State of the JOB), then it will store those results on the disk.



- **Interactive Operations on Spark RDD**
  This illustration shows interactive operations on Spark RDD. If different queries are run on the same set of data repeatedly, this particular data can be kept in memory for better execution times.



  By default, each transformed RDD may be recomputed each time you run an action on it. However, you may also **persist** an RDD in memory, in which case Spark will keep the elements around on the cluster for much faster access, the next time you query it. There is also support for persisting RDDs on disk, or replicated across multiple nodes.

VII. CONCLUSION

Apache Spark[7] is a powerful open source processing engine built around speed, ease of use, and sophisticated analytics. Since its release, Apache Spark has seen rapid adoption by enterprises across a wide range of industries. Internet powerhouses such as Yahoo, Baidu, Airbnb, eBay and Tencent, have eagerly deployed Spark at massive scale, collectively processing multiple petabytes of data on clusters of over 8,000 nodes. It has quickly become the largest open source community in big data[8], with over 1000 contributors from 250+ organizations. Spark provides a simple way to parallelize these applications across clusters, and hides the complexity of distributed systems programming, network communication, and fault tolerance. The system gives them enough control to monitor, inspect, and tune applications while allowing them to implement common tasks quickly. The modular nature of the API (based on passing distributed collections of objects) makes it easy to factor work into reusable libraries and test it locally.

REFERENCES

[1] Hadoop http://hadoop.apache.org & http://sortbenchmark.org/YahooHadoop.pdf
[2] Hadoop. http://hadoop.apache.org, 2009.
[3] HDFS (hadoop distributed file system) architecture. http://hadoop.apache.org/common/docs/current/hdfs design.html, 2009.
[4] http://spark.apache.org/
[5] http://spark.apache.org/examples.html
[6] www.safaribooksonline.com/library/view/learning-spark/9781449359034/ch01.html
[7] https://www.infoq.com/articles/apache-spark-introduction
[8] Jonathan Stuart Ward and Adam Barker "Undefined By Data: A Survey of Big Data Definitions" Stamford, CT: Gartner, 2012.
[9] Sri Hari Nallamala et all. - A Review on How YARN Overcomes MapReduce Limitations in Hadoop 2.0, International Journal of Advance Research in Engineering, Science & Technology (IJAREST), Vol.3, Issue.4, April 2016.
[10] Ahmed Eldawy, Mohamed F. Mokbel "A Demonstration of Spatial Hadoop: An Efficient MapReduce Framework for Spatial Data" Proceedings of the VLDB Endowment, Vol. 6, No. 12 Copyright 2013 VLDB Endowment 21508097/13/10.

AUTHOR'S BIOGRAPHY

Mr. V Srinivas Jonnalagadda* is working as an Assistant Professor at DVR & Dr. HS MIC College of Technology, Kanchikacherla. He received his M.Tech (CS) from KIET, Korangi, in the year 2012. He received his B.Tech (IT) from Sri Sarathi Institute Of Engineering & Technology, Nuzvid in the year 2010. His area of interest includes Computer networks and Big data.

Mr. P Srikanth** is working as an Assistant Professor at DVR & Dr.HS MIC College of Technology, Kanchikacherla. He received his M.Tech (CSE) from MIC COLLEGE in the year 2015. He received his B.Tech (IT) From Vikas College of Engineering & Technology, Nunna in the year 2012. His area of interest includes Computer Networks and Big Data Analytics.

Mr. Krishnamachari Thumati*** was Completed His B.Tech (IT) From Prasad Institute Of Technology and Sciences, Jaggaiahpet and M.Tech (Computer Science and Technology) From V.R. Siddhartha Engineering College, Vijayawada. Presently He was doing his services as an Assistant Professor at DVR & Dr.HS MIC College of Technology, Kanchikacherla, Krishna Dist., A.P. His interested domains are Big Data, Cyber Security.

Mr. Sri Hari Nallamala**** was completed his B.Tech (CSE) from JNT University Engineering College, Hyderabad and M.Tech (Computer Science) from JNT University Hyderabad. Now, he was pursuing his research work in Big Data (Data Mining) at K L University, Vijayawada. Totally, he has 6 years of teaching experience. He had published number of papers in reputed journals and presented papers at international & national conferences. He became as Reviewer for One International Journal and many on pipeline. Presently he was doing his services as an Assistant Professor at DVR & Dr.HS MIC College of Technology, Kanchikacherla, Krishna Dist., A.P. His interested domains are Data Mining, Cloud Computing, Image Processing, Robotics and Internet Technologies.