Volume 3, Issue 3, March-2016, pp. 134-142 ISSN (O): 2349-7084

International Journal of Computer Engineering In Research Trends

Available online at: www.ijcert.org

Clustering and Parallel Empowering Techniques for Hadoop File System

[1]K.Naga Maha Lakshmi , [2] A.Shiva Kumar

[1]*Asst Professor, Keshav Memorial Institute of Technology and Science, Narayanguda, Telangana, India,*

[2]*Asst Professor, Mahaveer Institute of Technology and Science, Bandlaguda, Telangana, India.*

Abstract—In the Big Data group, Apache Hadoop and Spark are gaining prominence in handling Big Data and analytics. Similarly MapReduce has been seen as one of the key empowering methodologies for taking care of large-scale query processing. These middleware are traditionally written with sockets and do not deliver best performance on datacenters with modern high performance networks. In this paper we investigate the characterizes of two file systems that support in-memory and heterogeneous storage, and discusses the impacts of these two architectures on the performance and fault tolerance of Hadoop MapReduce and Spark applications. We present a complete methodology for evaluating MapReduce and Spark workloads on top of in-memory file systems and provide insights about the interactions of different system components while running these workloads.

Keywords- Big Data, Big Data Analytics, MapReduce, Apache Spark

I.INTRODUCTION

The Hadoop framework is intended to give a dependable, shared storage and analysis base to the client group. The storage bit of the Hadoop framework is given by a distributed file system arrangement, for example, HDFS, while the analysis usefulness is displayed by MapReduce. A few different segments are a piece of the general Hadoop agreement suite. The MapReduce usefulness is planned as a device for profound data analysis and the change of large data sets. Hadoop empowers the clients to investigate/dissect complex data sets by using redid analysis scripts/commands. . In other words, by means of the redid MapReduce schedules, unstructured data sets can be distributed, broke down, and investigated crosswise over a huge number of shared-nothing preparing systems/Clusters/nodes. Hadoop's HDFS imitates the data onto different nodes to shield nature from any potential data-misfortune (to represent, if 1 Hadoop node gets fenced, there are no less than 2 different nodes holding the same data set).

Hadoop verses Relational Database Systems All through the IT group, there are numerous dialogs rotating around contrasting MapReduce with customary RDBMS arrangements. More or less, MapReduce and RDBMS systems reflects answers for totally diverse (estimated) IT handling situations and thus, a real correlation results into distinguishing the open doors and confinements of both arrangements in view of their particular functionalities and center ranges. Regardless, the data sets handled by customary database (RDBMS) arrangements are regularly much littler than the data pools used in a Hadoop situation (see Table 1). Consequently, unless an IT base procedures TB's or PB's of unstructured data in a very parallel environment, the announcement can be made that the execution of Hadoop executing MapReduce inquiries will be below average contrasted with SQL questions running against a (streamlined) social database.

Hadoop uses a beast power access strategy while RDBMS arrangements bank on streamlined getting to schedules, for example, files, and in addition read-ahead and compose behind procedures. Henceforth, Hadoop truly just exceeds expectations in situations that uncover a gigantic parallel handling base where the data is unstructured to the point where no RDBMS streamlining systems can be connected to support the execution of the inquiries.

Table 1: RDBMS and MapReduce Highlights

Feature/Application	Traditional RDBMS	MapReduce
Operating Data Size	Gigabytes +	Petabytes +
Access Pattern	Interactive and Batch	Batch
Update Scenarios	Repeated Read and Write	Write Once, Repeated Read
Data Structure	Static Schema	Dynamic Schema
Scaling Potential	Non-linear	Somewhat Close to linear

Hadoop is essentially intended to effectively handle expansive data volumes by connecting numerous merchandise systems together to function as a parallel element. Commercial enterprises are utilizing Hadoop widely to investigate their data sets. The reason is that Hadoop framework depends on a straightforward programming model (MapReduce) and it empowers a registering arrangement that is adaptable, adaptable, deficiency tolerant and practical. Here, the fundamental concern is to keep up velocity in handling huge datasets as far as holding up time in the middle of inquiries and holding up time to run the project. Flash was presented by Apache Programming Establishment for accelerating the Hadoop computational registering programming process. As against a typical conviction, Flash is not an adjusted rendition of Hadoop and is not, generally, reliant on Hadoop on the grounds that it has its own particular bunch administration. Hadoop is only one of the approaches to execute Flash. Spark utilizes Hadoop as a part of two ways – one is storage and second is handling. Since Spark has its own group administration calculation, it utilizes Hadoop for storage reason as it were.

II. DATA PROCESSING AND MANAGEMENT ON MODERN CLUSTERS

Big Data processing techniques analyze big data sets at terabyte or even petabyte scale. Processing, tackling arbitrary BI use cases. While real-time stream processing is performed on the most current slice of data for data profiling to pick outliers, fraud transaction detections, security monitoring, etc. The toughest task however is to do fast (low latency) or real-time ad-hoc analytics on a complete big data set. It practically means you need to scan terabytes (or even more) of data within seconds. This is only possible when data is processed with high parallelism. In this section we have presented how actually data processed and managed on various modern clusters which Substantial impact on designing and utilizing modern data management and processing systems in multiple tiers, the system consist of Front-end data accessing and serving (Online) eg: MySql, HBase Back-end data analytics (Offline) eg: HDFS, MapReduce, Spark

Big Data processing strategies break down large data sets at terabyte or even petabyte processing, handling subjective BI use cases. While real-time stream preparing is performed on the most current slice of data for data profiling to pick anomalies, misrepresentation exchange recognitions, security checking, and so on. The hardest undertaking however is to do quick (low latency) or ongoing ad-hoc examination on a complete big data set. It for all intents and purposes implies you have to output terabytes (or significantly more) of data inside of seconds. This is just conceivable when data is prepared with high parallelism. In this area we have introduced how really data handled and oversaw on different cutting edge bunches which Generous effect on planning and using advanced data administration and preparing systems in various levels, the system comprise of Front-end data getting to and serving (Online) eg: MySql, HBase Back-end data examination (Logged off) eg: HDFS, MapReduce, Spark

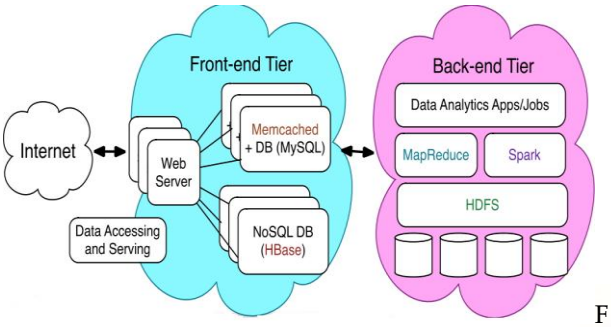

Fig 1. Data accessing and serving over internet

In the above fig data accessing and serving over internet where Front –end tier has web server which process through web server as MySql Queries or NoSql Queries later as Back-end tier data analytics will be done on MapReduce or Spark over HDFS .

2.1. Spark - An Example of Back-end Data Processing Middleware

An in-memory data-processing framework which performs Iterative machine learning jobs, Interactive data analytics. Scalable, communication and I/O intensive which performs Wide dependencies between Resilient Distributed Datasets (RDDs) – MapReduce-like shuffle operations to repartition RDDs – Sockets based communication

2.2. Cluster computing frameworks

MapReduce is one of the earliest and best known commodity cluster frameworks. MapReduce follows the functional programming model [8], and performs explicit synchronization across computational stages. MapReduce exposes a simple programming API in terms of map () and reduce () functions. Apache Hadoop [1] is a widely used open source implementation of MapReduce. The simplicity of MapReduce is attractive for users, but the framework has several limitations. Applications such as machine learning and graph analytics iteratively process the data, which means multiple rounds of computation are performed on the same data. In MapReduce, every job reads its input data, processes it, and then writes it back to HDFS. For the next job to consume the output of a previously run job, it has to repeat the read, process, and write cycle. For iterative algorithms, which want to read once, and iterate over the data many times, the MapReduce model poses a significant overhead. To overcome the above limitations of MapReduce, Spark [19] uses Resilient Distributed Datasets (RDDs) [19] which implement in-memory data structures used to cache intermediate data across a set of nodes. Since RDDs can be kept in memory, algorithms can iterate over RDD data many times very efficiently. Although MapReduce is designed for batch jobs, it is widely used for iterative jobs. On the other hand, Spark has been designed mainly for iterative jobs, but it is also used for batch jobs. This is because the new big data architecture brings multiple frameworks together working on the same data, which is already stored in HDFS [17]. We choose to compare these two frameworks due to their wide spread adoption in big data analytics. All the major Hadoop vendors such as IBM, Cloudera, Horton works, and MapR bundle both MapReduce and Spark with their Hadoop distributions

III.APACHE SPARK

Spark is another execution system. Like MapReduce, it works with the file system to disperse your information over the group, and process that information in parallel. Like MapReduce, Apache Spark is a fast and general engine for large-scale data processing. It is based on Hadoop MapReduce and it extends the MapReduce model to efficiently use it for more types of computations, which includes interactive queries and stream processing. 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.

3.1. Features of Apache Spark

3.1.1. Speed

Run programs up to 100x faster than Hadoop MapReduce in memory, or 10x faster on disk. Spark has an advanced DAG execution engine that supports cyclic data flow and in-memory computing.

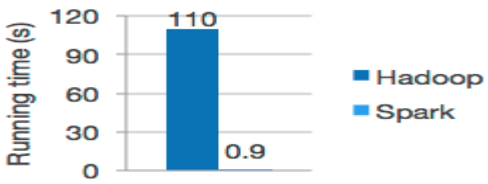

Fig2. Logistic regression in Hadoop and Spark

3.1.2. Ease of Use (Supports multiple languages)

Write applications quickly in Java, Scala, Python, R.Spark offers over 80 high-level operators that make it easy to build parallel apps. And you can use it interactively from the Scala, Python and R shells.

3.1.3. Advanced Analytics: Generality

Spark not only supports 'Map' and 'reduce'. It also supports SQL queries, Streaming data, Machine learning (ML), and Graph algorithms. Combine SQL, streaming, and complex analytics. Spark powers a stack of libraries including SQL and Data Frames, MLlib for machine learning, GraphX, and Spark Streaming. You can combine these libraries seamlessly in the same application.

3.2. Spark Built on Hadoop:

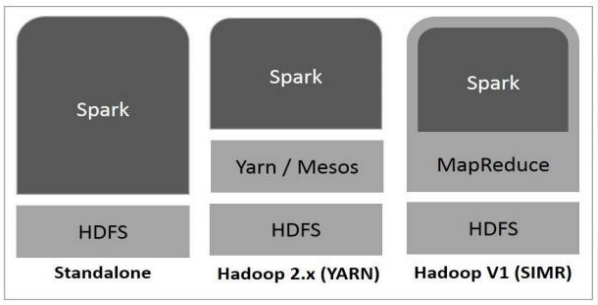

Fig 3. Spark built with Hadoop components.

3.2.1. Standalone: Spark Standalone deployment means Spark occupies the place on top of HDFS (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.

3.2.2. Hadoop Yarn: Hadoop Yarn 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.

3.2.3. 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.

3.3. Components of Spark

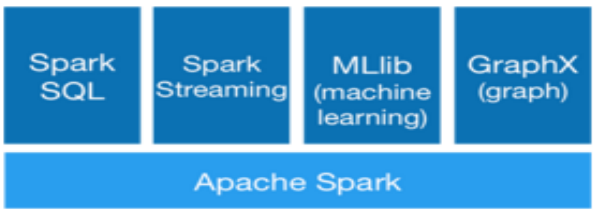

Fig 4. Illustration depicts the different components of Spark.

3.3.1. 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.

3.3.2. 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.

3.3.3.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.

3.3.4.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).

3.3.5.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.

3.4. Important of Resilient Distributed Datasets (RDD) in Apache Spark

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.

3.5. HDFS Architecture

Given below is the architecture of a Hadoop File System.

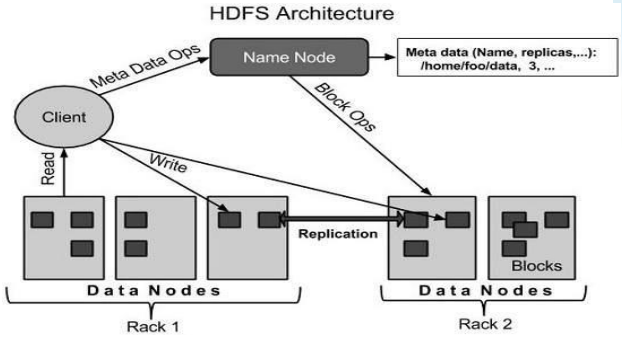

Fig 5. Architecture of a Hadoop File System

HDFS follows the master-slave architecture and it has the following elements.

Namenode: The namenode is the commodity hardware that contains the GNU/Linux operating system and the namenode software. It is software that can be run on commodity hardware. The system having the namenode acts as the master server and it does the following tasks: 1.Manages the file system namespace. 2. Regulates client's access to files. It also executes file system operations such as renaming, closing, and opening files and directories.

Datanode: The datanode is a commodity hardware having the GNU/Linux operating system and datanode software. For every node (Commodity hardware/System) in a cluster, there will be a datanode. These nodes manage the data storage of their system.

- Datanodes perform read-write operations on the file systems, as per client request.
- They also perform operations such as block creation, deletion, and replication according to the instructions of the namenode.

Block: Generally the user data is stored in the files of HDFS. The file in a file system will be divided into one or more segments and/or stored in individual data nodes. These file segments are called as blocks. In other words, the minimum amount of data that HDFS can read or write is called a Block. The default block size is 64MB, but it can be increased as per the need to change in HDFS configuration.

3.6. Goals of HDFS

Fault detection and recovery: Since HDFS includes a large number of commodity hardware, failure of components is frequent. Therefore HDFS should have mechanisms for quick and automatic fault detection and recovery.

Huge datasets: HDFS should have hundreds of nodes per cluster to manage the applications having huge datasets.

Hardware at data: A requested task can be done efficiently, when the computation takes place near the data. Especially where huge datasets are involved, it reduces the network traffic and increases the throughput.

IV.MAPREDUCE OVERVIEW

Apache Hadoop MapReduce is a framework for preparing large data sets in parallel over a Hadoop cluster. Data analysis utilizes a two stage outline lessen process. The employment setup supplies outline diminishes analysis capacities and the Hadoop system gives the planning, appropriation, and parallelization administrations. The top level unit of work in MapReduce is a vocation. An occupation as a rule has a guide and a diminish stage, however the lessen stage can be overlooked. For instance, consider a MapReduce work that checks the quantity of times every word is utilized over an arrangement of archives. The guide stage include the words every record, then the diminish stage totals the per-archive data into word checks spreading over the whole accumulation.

Amid the guide stage, the info data is separated into information parts for analysis by guide assignments running in parallel over the Hadoop cluster. Of course, the MapReduce structure gets information data from the Hadoop Appropriated Record Framework (HDFS). Utilizing the MarkLogic Connector for Hadoop empowers the system to get info data from a MarkLogic Server instance. For subtle elements, see Map task.

The reduce phase uses results from map tasks as input to a set of parallel reduce tasks. The reduce tasks consolidate the data into final results. By default, the MapReduce framework stores results in HDFS. Using the MarkLogic Connector for Hadoop enables the framework to store results in a MarkLogic Server instance. For details, see Reduce Task.

In spite of the fact that the lessen stage relies on upon yield from the guide stage, outline decrease preparing is not inexorably successive. That is, decrease undertakings can start when any guide errand finishes. It is redundant for all guide undertakings to finish before any lessen errand can start. MapReduce works on key-value pairs. Adroitly, a MapReduce work takes an arrangement of info key-value pairs and creates an arrangement of yield key-value pairs by going the data through guide and lessens capacities. The guide assignments create a middle of the road set of key-value pairs that the lessen errands utilizes as data. The chart underneath represents the movement from information key-value pairs to yield key-value pairs at an abnormal state

:

Though each set of key-value pairs is homogeneous, the key-value pairs in each step need not have the same type. For example, the key-value pairs in the input set (KV1) can be (string, string) pairs, with the map phase producing (string, integer) pairs as intermediate results (KV2), and the reduce phase producing (integer, string) pairs for the final results (KV3). See Example: Calculating Word Occurrences.

The keys in the map output pairs need not be unique. Between the map processing and the reduce processing, a shuffle step sorts all map output values with the same key into a single reduce input (key, value-list) pair, where the 'value' is a list of all values sharing the same key. Thus, the input to a reduce task is actually a set of (key, value-list) pairs.

The key and value types at each stage determine the interfaces to your map and reduce functions. Therefore, before coding a job, determine the data types needed at each stage in the map-reduce process. For example:

1. Choose the reduce output key and value types that best represents the desired outcome.

2. Choose the map input key and value types best suited to represent the input data from which to derive the final result.

3. Determine the transformation necessary to get from the map input to the reduce output, and choose the intermediate map output/reduce input key value type to match.

4. What is MapReduce? It is a part of the Hadoop framework that is responsible for processing large data sets with a parallel and distributed algorithm on a cluster. As the name suggests, the MapReduce algorithm contains two important tasks: Map and

Reduce. Map takes a set of data and converts it into another set of data, where individual elements are broken down into tuples (key/value pairs). On the other hand, Reduce takes the output from a map as an input and combines the data tuples into smaller set of tuples. In MapReduce, the data is distributed over the cluster and processed.

5. The difference in Spark is that it performs in-memory processing of data. This in-memory processing is a faster process as there is no time spent in moving the data/processes in and out of the disk, whereas MapReduce requires a lot of time to perform these input/output operations thereby increasing latency.

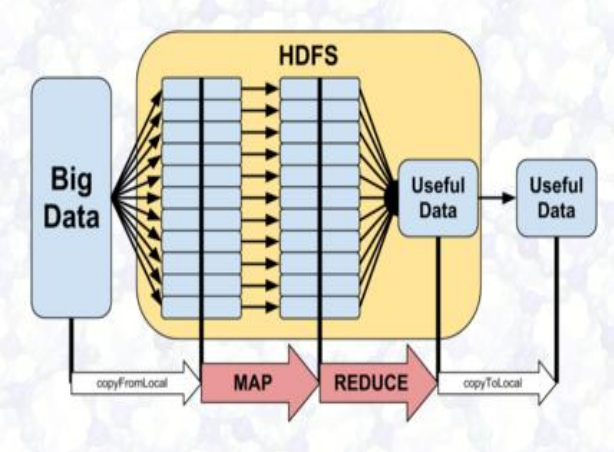

Fig 6. Operations of Mapreduce

5.1. Major difference among MapReduce and Spark

Real-Time Big Data Analysis:

Real-time data analysis means processing data generated by the real-time event streams coming in at the rate of millions of events per second, Twitter data for instance. The strength of Spark lies in its abilities to support streaming of data along with distributed processing. This is a useful combination that delivers near real-time processing of data. MapReduce is handicapped of such an advantage as it was designed to perform batch cum distributed processing on large amounts of data. Real-time data can still be processed on MapReduce but its speed is nowhere close to that of Spark.

Spark claims to process data 100x faster than MapReduce, while 10x faster with the disks.

Hadoop MapReduce	Apache Spark
Fast	100x faster than MapReduce
Batch Processing	Real-time Processing
Stores Data on Disk	Stores Data in Memory
Written in Java	Written in Scala

How Spark is compatible with Apache hadoop?

- Spark can keep running on Hadoop 2's YARN bunch supervisor,
- Spark can read any current Hadoop information.
- Hive, Pig and Mahout can keep running on Spark.
- Advanced Big Data Analytics is perplexing.
- Hadoop MapReduce (MR) works really well in the event that you can express your issue as a solitary MR work. Practically speaking, most issues don't fit flawlessly into a solitary MR work.
- MR is moderate for cutting edge Enormous Information Examination, for example, iterative preparing and storing of datasets which are appropriate to machine learning. Spark enhances MapReduce by uprooting the need to compose information to plate between steps.
- Need to coordinate numerous different apparatuses for cutting edge Big Data Analytics for Questions, Gushing Examination, Machine Learning and Chart

5.2. Key capabilities of Spark

- Fast parallel data processing as intermediate data is stored in memory and only persisted to disc if needed.
- Apache Spark provides higher level abstraction and generalization of MapReduce. Over 80 high level built-in operators. Besides MapReduce, Spark supports SQL queries, streaming data, and complex analytics such as machine learning and graph algorithms out-of-the-box.
- MapReduce is limited to batch processing. Spark lets you write streaming and batch-mode

applications with very similar logic and APIs instead of using different tools.

- Apache Spark provides a set of composable building blocks for writing in Scala, Java or Python,concise queries and data flows. This makes developers highly productive.
- It is possible to build a single data workflow that leverages, streaming, batch, sql and machine learning for example.
- Spark runs on Hadoop, Mesos, standalone, or in the cloud. It can access diverse data sources including HDFS, Cassandra, HBase, S3.

VI.PERFORMANCE

Iterative Machine Learning Algorithms:

- Almost all machine learning algorithms work iteratively. As we have seen earlier, iterative algorithms involve I/O bottlenecks in the MapReduce implementations. MapReduce uses coarse-grained tasks (task-level parallelism) that are too heavy for iterative algorithms. Spark with the help of Mesos – a distributed system kernel, caches the intermediate dataset after each iteration and runs multiple iterations on this cached dataset which reduces the I/O and helps to run the algorithm faster in a fault tolerant manner.
- Spark has a built-in scalable machine learning library called MLlib which contains high-quality algorithms that leverages iterations and yields better results than one pass approximations sometimes used on MapReduce.

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.

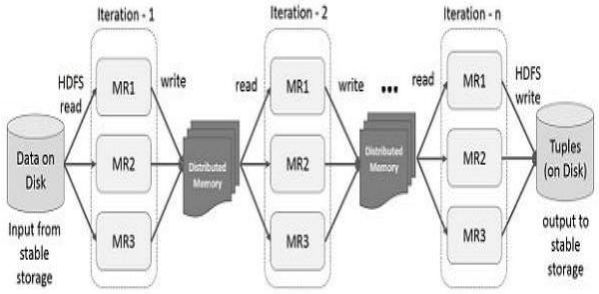

Figure 7: Iterative operations on Spark RDD

If different queries are run on the same set of data repeatedly, this particular data can be kept in memory for better execution times

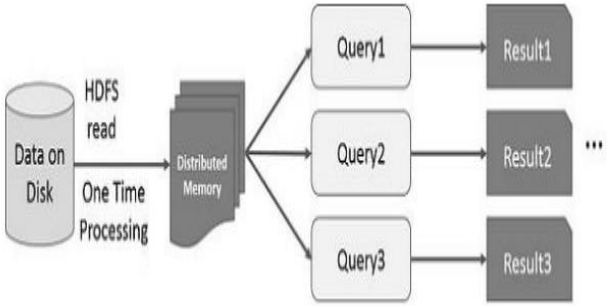

Figure 8: Interactive operations on Spark RDD

In a Big Data connection, each of these cycles can be exceptionally burdensome, with every test cycle, for instance, being hours long. While there are different routes systems to mitigate this issue, one of the best is to just run your project quick. On account of the execution advantages of Spark, the improvement lifecycle can be really abbreviated simply because of the way that the test/investigate cycles are much shorter.

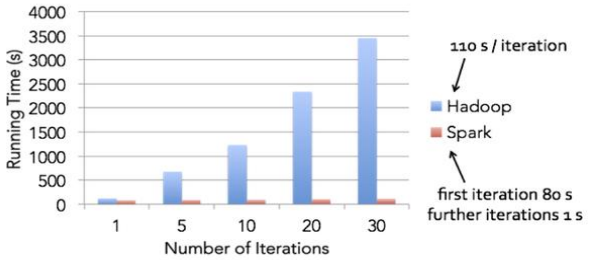

Figure 9.Performance of Spark

Here are results from a survey taken on Spark by Typesafe to better understand the trends and growing demand for Spark.

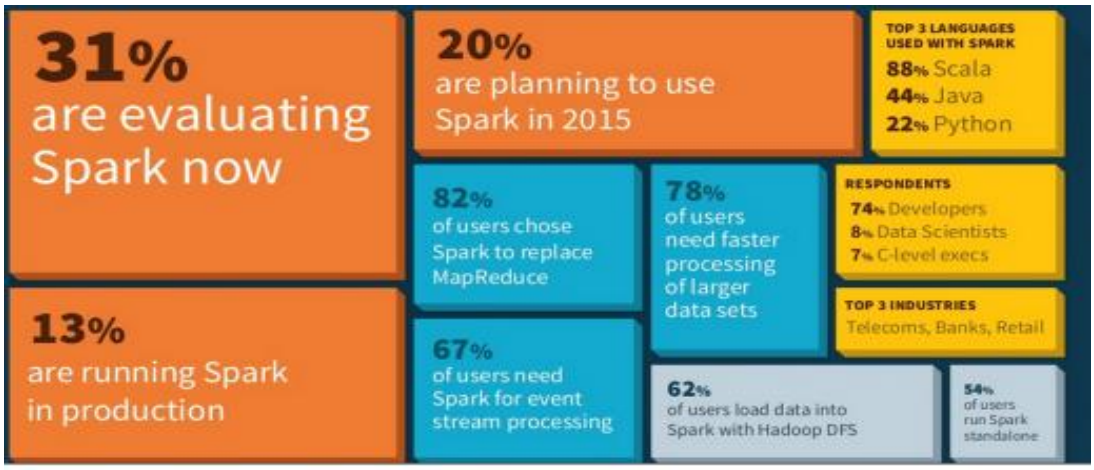

Fig 10. Apache Spark survey Report

VII. CONCLUSIONS

In this paper we looked into the MapReduce hindrance in meeting the reliably extending enlisting solicitations of Big Data. In this papers we focused on substitution of MapReduce technique, one of the key engaging approaches for dealing with Big Data asks for by strategy for significantly parallel get ready on innumerable nodes. Issues and troubles MapReduce faces while overseeing Tremendous Data are leads with Apache Spark it has practically identical levels of security, sensibility, and flexibility as MapReduce, and should be correspondingly joined with the straggling leftovers of the advancements that incorporate the ceaselessly developing Hadoop platforms.

REFERENCES:

[1] P. Zadrozny and R. Kodali, Big Data Analytics using Splunk, Berkeley, CA, USA: Apress, 2013.

[2] F. Ohlhorst, Big Data Analytics: Turning Big Data into Big Money, Hoboken, N.J, USA: Wiley, 2013.

[3] J. Dean and S. Ghemawat, "MapReduce: Simplified data processing on large clusters," Commun ACM, 51(1), pp. 107-113, 2008.

[4] Apache Hadoop, http://hadoop.apache.org.

[5] F. Li, B. C. Ooi, M. T. Özsu and S. Wu, "Distributed data management using MapReduce," ACM Computing Surveys, 46(3), pp. 1-42, 2014.

[6] C. Doulkeridis and K. Nørvåg, "A survey of large-scale analytical query processing in MapReduce," The VLDB Journal, pp. 1-26, 2013.

[7] P. Bhatotia, A. Wieder, R. Rodrigues, U. A. Acar and R. Pasquin, "Incoop: MapReduce for incremental computations," Proc. of the 2nd ACM Symposium on Cloud Computing, 2011.

