IJCRT.ORG ISSN : 2320-2882

INTERNATIONAL JOURNAL OF CREATIVE RESEARCH THOUGHTS (IJCRT)

An International Open Access, Peer-reviewed, Refereed Journal

Big Data Frameworks

Apache Hadoop And Apache Spark

[1]Omkar Bhambe, [2]Anurag Sandbhor, [3]Dr. Manjusha Tatiya

[1]Student, [2]Student, [3]Head of Department,
[1]Department of Artificial Intelligence and Data Science,
[1]Indira College of Engineering and Management, Pune, India

Abstract: This review paper gives information on Big Data and Hadoop and Spark, along with the other technologies. It shows a study on MapReduce and Hadoop distributed File System (HDFS)[3] which are important in Hadoop as well as Resilient Distributed Datasets(RDDS) [6], Spark SQL [7], MLlib [5] which areimportant for Apache Spark. Along with that other Big Tools such as Apache Flume [3], Apache Kafka [2], Apache Sqoop, [3] etc. It also shows the difference between these two open-source frameworks and what future technologies are supposed to be integrated so that they can be more efficient and speed. What they lack and how these can be solved.

Index Terms **-** Apache Hadoop, Apache Spark, HDFS, MapReduce, Apache Kafka, Apache Flume, Apache Sqoop, RDDS, Spark SQL, MLLib.

I. INTRODUCTION

Big data analysis refers to the process of examining large and complex datasets that are beyond the capability of traditional

data-processing tools [3]. As industries generate massive volumes of data from various sources—such as social media, sensors, and transaction logs—efficient analysis becomes critical. Big data is characterized by its five V's: volume, velocity, variety, veracity, and value [8]. To extract meaningful insights, big data analysis relies on advanced algorithms, machine learning techniques, and distributed computing frameworks like Hadoop and Spark [9]. Industrially big data analysis span multiple sectors, including healthcare, finance, retail, and manufacturing. In healthcare, for example, big data analytics can be used to predict disease outbreaks and personalize treatment plans [2]. In finance, it helps in fraud detection and risk management [2]. Retailers can use customer behavior data to optimize marketing strategies and inventory management [1].

II. INITIAL STAGES OF BIG DATA

The emergence of big data can be traced back to the early 2000s when organizations began generating unprecedented amounts of data from various sources such as digital transactions, social media, sensors, and web logs [3]. Traditional data processing systems could no longer manage this scale, and thus began the need for specialized tools and frameworks [4]. Around 2005, with the introduction of various social media like Facebook, YouTube and other online services, people started realizing how much data is being generated by the users [9]. And in the same year an open source framework 'Hadoop' [3] was created to store and analyze these data which is was most necessary at the moment and till today it is the necessity. Hadoop made a big break through due to its functionalities to store and processing of big data which was cheaper as compared to the conventional file processing and NoSQL and relational databases [1]. In next few years, the volume of the big data skyrocketed [10]. And at this point no only humans were generating the data, devices and systems also started generating the data which boomed the generation of the data to an higher extend which leads to

the creation of [2] better frameworks for storing and analyzing these large amount of data [3].

III. TECHNOLOGIES USED IN HADOOP

Hadoop is a open source Framework which used to store and analyze the big data. It is one of the first frameworks which was came into existence with the Big Data [2]. As we know that Big Data is a very large amount of data and it is difficult and very costly to process and analyze this data using conventional file processing, which was not conventional and beneficial industrially [9]. So, developers (Doug Cutting and Mike Cafarella in 2005) [11] of Hadoop build this system which was very impactful act the time and helped the industry to develop their research into the field of big data [2], which helped industrialists and researchers impactfully. It became easier to store this large data and analyze this data more efficiently. It helps to maintain this data in efficient and fault-tolerant
manner [3]. Hadoop has two main components HDFS (Hadoop Distributed File System) and Map Reduce along with this there are some tools which are based on Hadoop such as Apache Pig, Apache HBase, Apache Hive, Apache Sqoop, Apache Flume, and Apache Zookeeper [9].

1. Hadoop Ecosystem [9]:

The Hadoop Ecosystem is a suite of open-source tools and frameworks designed to store, process, and analyze large-scale datasets in a distributed computing environment [9]. It is built around the Hadoop framework, which primarily includes the Hadoop Distributed File System (HDFS) [9] for storage and the MapReduce programming model for data processing [3]. The ecosystem also consists of several other components that work together to handle various aspects of big data processing and management.

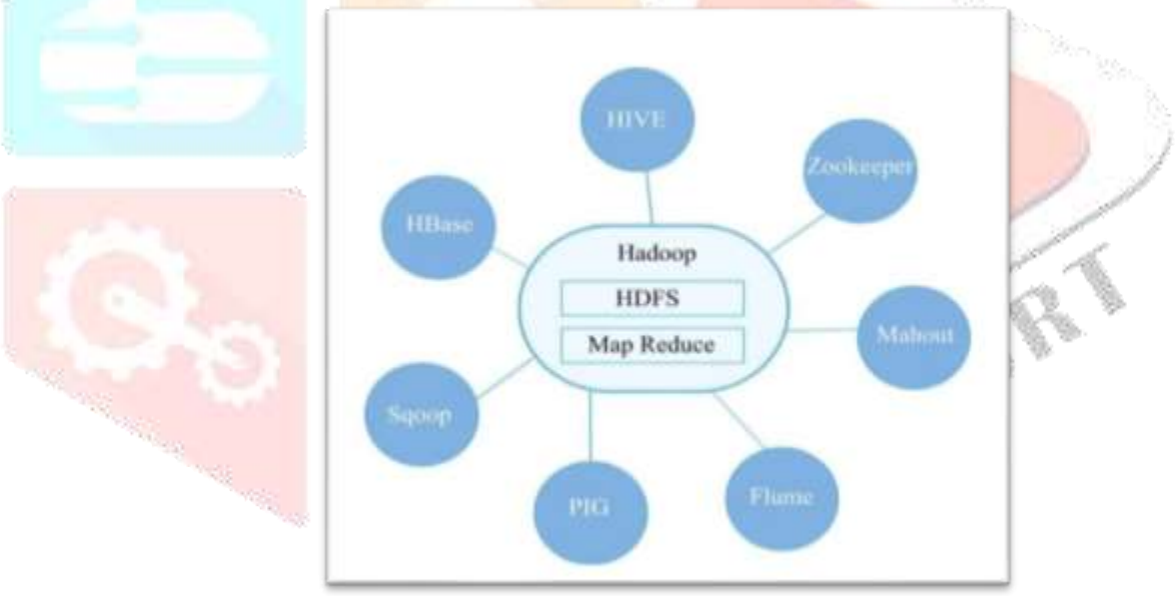

Fig. 1 Hadoop Ecosystem [9]

Let's see the components which are mentioned in the above architecture of the ecosystem of the Hadoop.

2. Hadoop Distributed File System:

Hadoop Distributed File System (HDFS) [3] it is a primary data storage mechanism used by Hadoop system. As shown in the Fig. 1, there is a NameNode [3] and DataNode [3] architecture which provides high performance data accessibility across highly
scalable Hadoop Cluster. HDFS is a key part in the Hadoop ecosystem [9].

NameNode is a master node which controls all the data nodes and it contains meta data [3]. It also helps in managing all the operations such as read, write, update, etc [3].

Data nodes this are also known as slave nodes, all the file and data operations performed on these nodes are actually stored on these nodes and are decided by the name nodes [3].

There is also a secondary name node which works as the back for the name node [3]. Name node is the master so it becomes very important that there should be a back of it. So if in any situation if there is occurrence of failure to name node, then secondary
name node will be used.

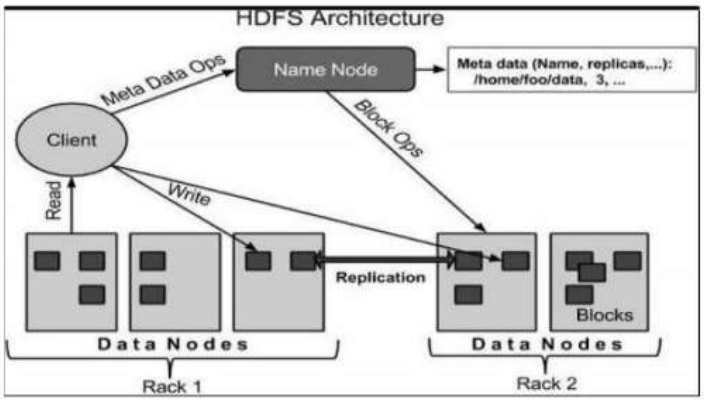

Fig. 2 Hadoop Distributed File System Architecture [3]

3. MapReduce:

MapReduce is a distributed computing framework that allows for the efficient processing of large data sets across multiple computing nodes [2]. Originally developed by Google [2], MapReduce has since been implemented in open-source environments like Hadoop, where it plays a critical role in handling big data [2]. The core idea of MapReduce is to break down a large task into smaller, manageable units, which are then processed independently [3]. These units are mapped to specific tasks and, after processing, are reduced into final outputs [2].

MapReduce contains two main stages: Map and Reduce. The Map stage processes the input data and breaks it down into key-value pairs [2]. This intermediate data is then passed to the Reduce stage, which aggregates and processes these key-value pairs to generate the final output [2]. The simplicity of this model, combined with its ability to scale across thousands of machines, makes it a powerful tool for distributed data processing [9].

It operates on Four main or we can say core components:

a. Input: It is the raw data that is divided into smaller chunks and distributed to the Mappers [9].

b. Mapper: The component that processes each chunk of data, breaking it down into key-value pairs for further processing [9].

c. Reducer: The reducer takes the output from the mappers and combines it, processing the aggregated data into a final result [9].

d. Output: The final processed data, produced after the reduce stage, is outputted as the aggregated result [9].

By breaking a large process into smaller tasks into smaller sub-tasks that runs concurrently, MapReduce ensures efficient processing and easy scalability, which is why it is widely used in big data applications.

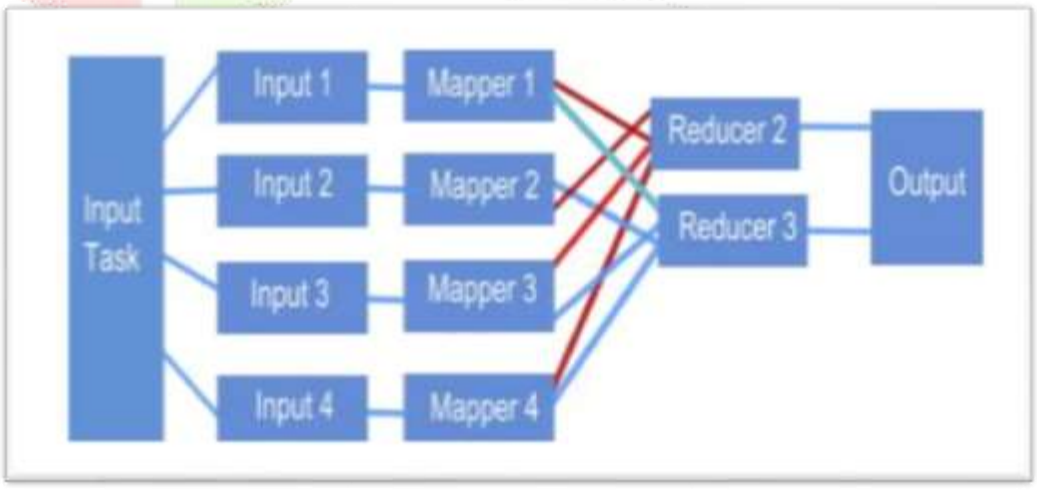

Fig. 3 MapReduce Architecture [9]

4. Sqoop

Sqoop is a tool for transferring data between Hadoop and relational databases [3]. It can import data from structured databases like MySQL or Oracle into HDFS, and also export data back from HDFS to those databases [3].

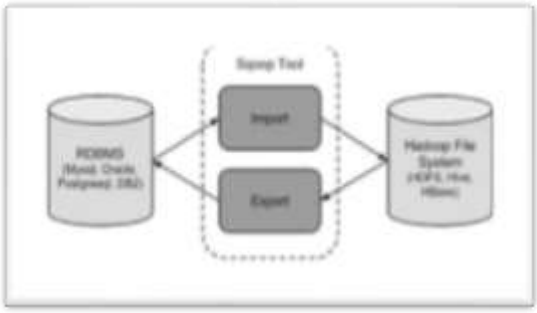

Fig. 4 Sqoop Architecture [3]

5. Hive

Hive is a data warehousing tool built on top of Hadoop that allows users to query and manage large datasets stored in HDFS using a SQL-like query language called HiveQL [3]. Hive abstracts the 4 complexity of MapReduce jobs and is popular among
users who are familiar with SQL [3].

6. HBase

HBase is a NoSQL database that runs on top of HDFS and allows for random, real-time read/write access to large datasets [3]. It is particularly useful for applications that require fast access to large amounts of sparse data, and is commonly used when low-latency access to data is required [3].

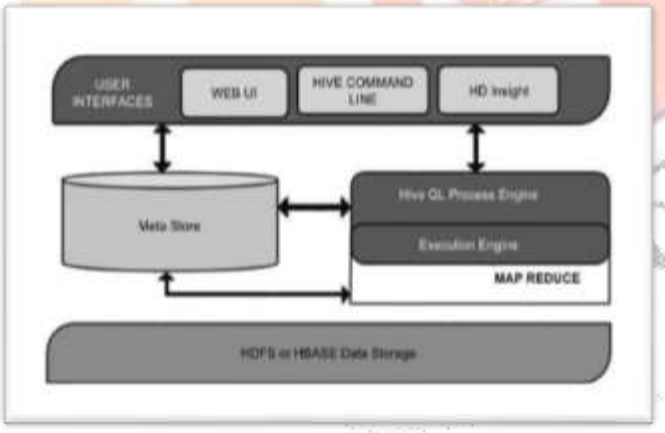

Fig. 5 HBase Architecture [3]

7. Pig

Pig is a high-level data flow language that simplifies writing complex MapReduce jobs. It uses a scripting language called Pig [3] Latin to express data transformations such as filtering, aggregation, and joining. Pig is particularly useful for data engineers
who need to process large, unstructured data sets [3].

8. Flume

Flume is a tool for collecting, aggregating, and transporting large amounts of streaming data into HDFS [3]. It is often used for ingesting log data from various sources such as servers or applications [3].

9. Mahout

Mahout is a machine learning library built on top of Hadoop [3]. It provides a range of scalable algorithms for tasks like classification, clustering, and collaborative filtering, allowing for data-driven decision-making using big data [3].

10. Zookeeper

Zookeeper is a centralized service for maintaining configuration information, naming, and synchronization across distributed systems [3]. It is essential for coordinating distributed applications in the Hadoop Ecosystem [3].

IV. ADVANTAGES AND LIMITATIONS OF HADOOP COMPARED TO OTHER TECHNOLOGIES AND TOOLS.

These are the advantages and limitations we saw through our study. Throughout this study we learn so many new facts and industrial malfunctions which were saved by all these technologies in the field of Big Data. We got to know that how much it was necessary to study this large amount of data.

Along with these we got know how Hadoop was more powerful and advantageous then that of the other technologies but still it has limitations as well, which were necessary to be taken care off and both advantages and limitations are listed below:

Advantages of Hadoop

i. Scalability: Hadoop is designed to scale horizontally by adding more nodes to a cluster [8]. This allows it to handle massive amounts of data without being limited by the capabilities of a single machine [3]. Compared to Apache Kafka, which is more focused on real-time data streaming, Hadoop is better suited for large-scale batch processing [11].

ii. Cost-Effectiveness: Hadoop uses commodity hardware, which makes it cheaper to implement compared to proprietary systems or high-end servers [11]. Tools like Spark, while faster, may require more expensive memory-based hardware for real-time analytics [5].

iii. Distributed Data Storage: Hadoop's Hadoop Distributed File System (HDFS) [3] efficiently stores and manages large datasets across distributed clusters [3]. This makes Hadoop ideal for managing vast amount of unstructured and semi-structured data [1], whereas tools like Apache Flume are more specialized in streaming data and don't offer the same level of long-term distributed storage [3].

iv. Fault Tolerance: HDFS is designed with built-in fault tolerance [12]. Data is replicated across nodes, ensuring that if one node fails, the data can still be recovered from other nodes [4]. While tools like Apache Kafka also have fault tolerance, Hadoop's replication system is more focused on long-term storage and recovery for large datasets [3].

v. Open-Source Flexibility: Being open-source, Hadoop provides flexibility and cost savings compared to some enterprise-level solutions [2]. Its ecosystem also integrates well with a variety of other tools, including Spark and Kafka, which allows users to build highly customized big data solutions**.**

Limitations of Hadoop:

i. Slow Performance: One of the main drawbacks of Hadoop is its relatively slow performance [1], especially when compared to Apache Spark. Hadoop relies heavily on disk-based storage for processing [3], which introduces latency. Spark's in-memory processing is much faster, particularly for iterative algorithms and real-time analytics.

ii. High Latency: Hadoop is optimized for batch processing rather than real-time data streaming [2]. This makes it less suitable for use cases requiring immediate data insights [3]. On the other hand, Apache Kafka and Flume are designed specifically for real-time data ingestion and processing [3], offering lower latency in streaming applications.

iii. Memory – Insensitive Requirements: While Hadoop can work on commodity hardware, its performance can suffer without enough memory, particularly when handling complex data transformations [9]. Apache Spark, in contrast, is optimized for memory-intensive tasks and can perform up to 100 times faster for certain operations [2].

iv. Lack of Real-Time Processing: Hadoop's MapReduce model is not ideal for real-time analytics [2], where tools like Apache Spark and Apache Kafka outperform Hadoop. Spark's real-time processing capabilities with Spark Streaming [5] and Kafka's low-latency data streaming [3] are more appropriate for scenarios requiring instant insights.

v. Security Concerns: Hadoop's security model is relatively basic, with initial versions lacking sophisticated authentication and encryption features [1].While modern versions have improved security through Kerberos [1], it is still not as advanced as some enterprise-grade solutions. In comparison, tools like Kafka have more robust security features, especially in data transmission.

V. INTRODUCTION TO APACHE SPARK

Apache Spark is an open-source cluster-computing framework [1]. Apache Spark was introduced by the Apache Software Foundation to enhance the speed of the Hadoop computational process [7]. Contrary to popular belief, Spark is not a modified version of Hadoop and doesn't rely entirely on it, as it has its own system for managing clusters [7]. Hadoop is just one option for using Spark, mainly for storage and, in some cases, for processing tasks [7]. Spark's primary, advantage is its in-memory cluster computing capability,which significantly boosts the speed of application processing [7].

Spark is versatile, handling various types of workloads, including batch processing, iterative algorithms, interactive queries, and real-time data streaming [5]. By supporting all of these workloads within a single system, Spark simplifies the management process, eliminating the need for multiple specialized tools [6].

Initially developed as a sub-project of Hadoop in 2009 at UC Berkeley's AMPLab by Matei Zaharia [5], Spark was made open-source in 2010 under a BSD license. In 2013 [5], it was donated to the Apache Software Foundation, and by February 2014, Spark became a top-level Apache project [7].

VI. IS SPARK REPLACING HADOOP?

Hadoop is a parallel data processing framework traditional used for running MapReduce jobs, which can take minutes or even hours to complete [7]. In contrast, Spark is designed to run on top of Hadoop and offers an alternative to the traditional batch MapReduce model [7]. Spark enables real-time stream data processing and fast, interactive queries thatcan be completed in seconds [6].

Hadoop is versatile, as it supports both the traditional MapReduce jobs and Spark. It's best to view Hadoop as ageneral-purpose framework that can accommodate different data processing models [2]. Spark should be seen as an alternative to Hadoop's MapReduce, not a replacement for Hadoop itself [7].

As we are discussing Apache Spark, we had also gained some knowledge of how Spark is different from Hadoop. Which is our main objective to study how these two technologies are independent and are different from each other, how they can be developed and what future integrations does they need.

So, Let's dive deep into spark let's understand the features of Apache Spark and how are they different and more better

or limited as compared to Hadoop.

VII. FEATURES OF APACHE SPARK

a. Speed

Spark enables applications to run significantly faster on a Hadoop cluster, achieving up to 100 times the speed in memory and up to 10 times faster when operating on disk. This is made possible by minimizing the number of disk read/write operations. Spark stores intermediate processing data in memory and only writes it to disk when absolutely necessary. It relies on the concept of a **Resilient Distributed Dataset (RDD),** which allows data to be transparently held in memory, reducing the need for frequent disk access, one of the primary time-consuming aspects of data processing.

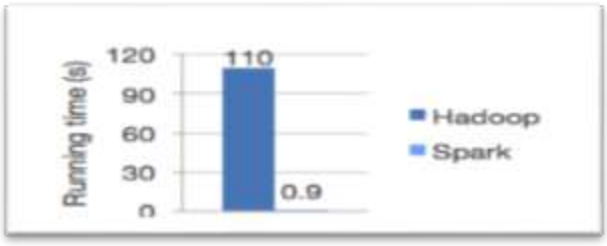

Fig. 7 Running Time Required by both the frameworks [7]

Resilient Distributed Datasets (RDDs) are the core data structure in Apache Spark [6]. They represent a fault-tolerant collection of data elements that can be processed in parallel across a cluster of machines [4]. RDDs provide a flexible and efficient way to perform transformations and actions on large datasets [5].

RDDs are resilient because they automatically recover from faults, such as node failures, by keeping track of the lineage, or the sequence of transformations applied to the data [7]. This allows Spark to recompute lost partitions in case of failure, without requiring the data to be replicated.

RDDs support two types of operations: transformations and actions [6]. Transformations (like map, filter, and groupBy) create new RDDs from existing ones, while actions (like count, collect, and saveAsTextFile)

[5] trigger computation and return a result or save the data. RDDs use lazy evaluation, meaning transformations are not executed until an action is performed, which helps optimize execution plans [4].
RDDs can be created from data in external storage systems like HDFS, Cassandra, or local file systems, or by parallelizing existing collections [7]. They also offer persistence options, allowing users to cache datasets in memory for faster access during iterative operations, which is useful for machine learning algorithms and other repeated data processes [5].

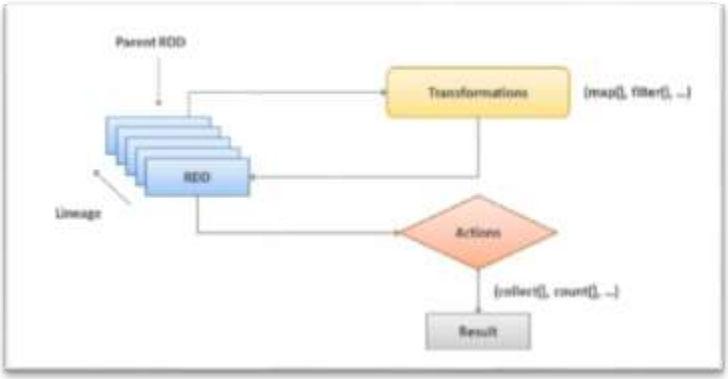

Fig. 8 Work Flow of Spark using RDDs [6]

b. Ease to Use

Spark's DataFrame and Dataset APIs provide optimized ways to work with structured and semi-structured data [6]. These APIs offer a higher level of abstraction than RDDs, allowing users to apply SQL-like [7] operations on large datasets. DataFrames are designed for efficiency, leveraging Spark's Catalyst optimizer, while Datasets add type-safety to these operations [4].
Apache Spark allows developers to quickly build applications using familiar programming languages such as **Java, Scala, or Python** [7]. This makes it easier for developers to create and run parallel applications. Spark also provides over 80 built-in high-level operators [7], simplifying complex data processing tasks. Additionally, it can be used interactively to query data directly from the shell [7].

For example here is an simple word count program using 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) [7]
```

Apart from basic operations like map and reduce, Spark supports more advanced functionalities such as SQL queries, real-time
data streaming, machine learning, and graph processing [7]. All these capabilities can be seamlessly integrated into a single workflow, making Spark a versatile tool for various types of data analytics [7]

c. Catalyst Optimizer

The Catalyst Optimizer is a powerful query optimization engine in Spark SQL [6]. It transforms queries into an efficient execution plan using rules for logical and physical query transformations [7]. This optimization results in faster query execution, making Spark SQL highly efficient for complex analytics [5].
It is also considered as one of the core components of spark responsible for optimizing queries to improve execution performance [7]. It is part of the Spark SQL module and plays a crucial role in transforming and refining the query execution plan [5]. The optimizer uses a combination of rule-based and cost-based optimization techniques to analyze query logic and generate an efficient execution plan [4].

d. MLib

Apache Spark facilitates the development of large-scale machine learning algorithms, making it ideal for scenarios where data parallelism or model parallelism is important [7]. Spark's core is specifically designed to handle iterative computations efficiently, which is a key requirement for many machine learning tasks [5]. These algorithms typically involve repeated processing steps, and Spark's architecture makes this process faster and more streamlined [7].
Building machine learning algorithms and pipelines for real-world applications usually involves a few critical steps: feature extraction, data transformations, model training, evaluation, and tuning [5]. To

simplify these tasks, Spark offers **MLlib**, its distributed machine learning library, designed to help developers easily implement and design machine learning workflows [6].

MLlib is divided into two core packages: **spark.mllib** and **spark.ml.** [7] The spark.mllib package is based on **RDDs** (Resilient Distributed Datasets) [7], while spark.ml is built on top of DataFrames, offering a more flexible and higher-level API [7]. Both packages come with tools to perform key machine learning operations like feature transformation, model training, and evaluation [7].

The **spark.ml** package also provides a user-friendly Pipelines API, which streamlines the process of building, debugging, and tuning machine learning workflows [5]. On the other hand, **spark.mllib** includes foundational tools like linear algebra libraries, statistical functions, and utilities that are essential for machine learning [7].

In essence, Spark's **MLlib** is designed to simplify the process of building scalable machine learning systems, whether for feature

engineering or model evaluation, allowing for efficient large-scale processing [6].

e. Graphx

GraphX is a powerful library within Apache Spark designed for large-scale graph analytics [6]. It combines the strengths of traditional graph-parallel systems with Spark's data-parallel framework, offering a robust toolset for graph-based computations [6]. By extending the RDD (Resilient Distributed Dataset) API, GraphX introduces an efficient abstraction to represent and manipulate graph-oriented data [6].

GraphX provides users with various graph transformations, commonly used graph algorithms, and a range of graph-building utilities [6]. It also includes a variant of the **Pregel API**, which is tailored for graph-parallel computations, allowing efficient processing of large graphs [6]. A unique aspect of GraphX is its ability to integrate both data transformations from Spark Core

and graph transformations, enabling users to seamlessly build complete graph analysis workflows [6].

With GraphX, you can create end-to-end pipelines that handle both data and graph computations, offering a unified approach

to large-scale graph analytics [6].

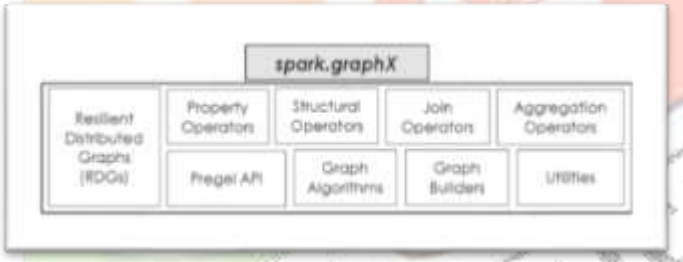

Fig. 8 Key Features Graphx [6]

f. Structured Streaming

Most traditional stream processing systems operate by handling records one at a time, following a continuous operator model [5].

While this approach works well for smaller scales, it encounters challenges when dealing with large-scale and real-time data analysis [5]. To address these issues, Spark Streaming employs a micro-batch architecture [6]. In this model, a stream is viewed

as a series of small batches of data, and the streaming computations are carried out through successive batch processes [6].

To implement this micro-batch architecture, Spark Streaming includes several key components for stream processing [6]. The Streaming Context [6] serves as the primary entry point for all streaming functionalities. When creating a Streaming Context, a

crucial parameter known as the batch interval must be set according to the latency needs of the application and the available resources of the cluster [6].

DStream (Discretized Stream) [6] is the fundamental programming abstraction in Spark Streaming, enabling micro-batch processing [6]. Various data sources can be connected to Spark Streaming, representing different types of streaming input [6]. A receiver acts as an interface between a data source and Spark Streaming, acquiring data and temporarily storing it in Spark's memory for processing later [5]. Additionally, a scheduler provides a listener interface that allows monitoring of ongoing streaming computations, including the status of receivers and processing times [6].

Moreover, Spark Streaming supports a wide range of transformations and output operations, along with various utilities for effective stream processing [5].

VIII. ADVANTAGES AND LIMITATIONS OF SPARK

As spark is developed from the Hadoop. It showcases a great deal of fast processing, memory boosting efficient user experience and model training mechanism it still has some limitations as compared to other technologies and Hadoop, so let's see all this advantages and limitatios.

Advantages of Spark:

1. Speed: Spark performs in-memory computation, making it much faster than traditional MapReduce [2]. It can handle both batch processing [1] and real-time data streaming, which significantly reduces the time required for complex computations [6]. It is especially faster when working with iterative machine learning algorithms or graph computations [5].
2. Ease of Use: Spark provides high-level APIs in languages like Java, Scala, Python, and R [7], making it easier for developers and data scientists to work with, compared to frameworks like Hadoop, which require writing complex MapReduce jobs [7]. The simplicity of Spark's APIs allows easier integration into various data workflows.
3. Unified Engine: Spark can handle a wide variety of tasks like batch processing, stream processing, interactive queries, machine learning, and graph computation in one unified framework [6]. This makes it versatile for a range of big data problems.
4. Real-time Data Processing: With Spark Streaming, Spark is equipped for real-time stream processing, unlike Hadoop MapReduce, which is primarily for batch processing [6]. Compared to tools like Apache Kafka and Apache Flume, which are primarily focused on message streaming and ingestion, Spark extends the capabilities by adding processing logic on top of
streams [3].
5. Rich Ecosystem: Spark has built-in libraries for SQL (Spark SQL), machine learning (MLlib), graph processing (GraphX), and streaming (Spark Streaming) [5]. This offers a comprehensive ecosystem for big data processing. It can work with different data sources such as HDFS, Hive, HBase, Cassandra, and more [6].
6. Fault Tolerance: Spark provides fault tolerance through lineage information, making it easier to recover lost data during processing, [4] as it reconstructs the data from previous steps. It ensures data recovery in case of failure, similar to Hadoop but with less overhead [6].

Limitations of Spark:

1. Memory Consumption: Spark's in-memory processing can lead to high memory consumption [1]. Large datasets may require distributed computing clusters with significant memory resources, which could increase infrastructure costs [3]. For data too large to fit into memory, Spark may fall back to disk-based processing, significantly reducing its speed advantages [6].
2. Complexity with Real-Time Processing: While Spark Streaming enables real-time data processing [1], it is notas natively focused on message queuing and event- based systems as Kafka or Flume [4]. Hence, it requires extra configuration to manage real-time processing effectively [7]. Tools like Kafka are more optimized for high-throughput message handling with low latency [3].
3. Expensive Resource Usage: Due to its heavy reliance on memory, Spark can sometimes be more resource- intensive than Hadoop [1], which utilizes disk-based storage, making it more cost-effective for certain large- scale, batch-only operations [1].
4. No Built-in Storage Layer: Unlike Hadoop, which comes with HDFS as its storage [10], Spark does not have its own distributed storage system. It relies on other storage systems like HDFS, Amazon S3, or Apache HBase for data persistence [5].
5. Less Mature Ecosystem: While Spark has a strong community and ecosystem [6], it is relatively newer compared to Hadoop. Some tools and integrations may not be as mature or stable as those available in the Hadoop ecosystem [1].
6. Latency for Small Data Sets: For small datasets, the overhead of Spark's in-memory processing can lead to higher latency compared to simpler solutions [1]. In some cases, traditional batch processing frameworks may perform better [1].

IX. COMPARATIVE DIFFERENCE BETWEEN HADOOP AND SPARK

As both Hadoop and Spark are having their own technologies and features. Their still differences between them through their development, features and functionalites they provide to the user while using these. As we can see from the above study that Hadoop is one of the first Big Data Open-Source Developed by Apache Foundation [1] and based on it many tools and Frameworks were developed. Apache Spark is also one of those Open-Source Frameworks Developed using Hadoop but it is not

dependent on Hadoop like Hive and HBase [7]. It is completely independent but can be integrated or used with Hadoop which

gives it a upper hand than the other Frameworks [7]. Below there are the work flow structures of both Hadoop and Apache Spark.

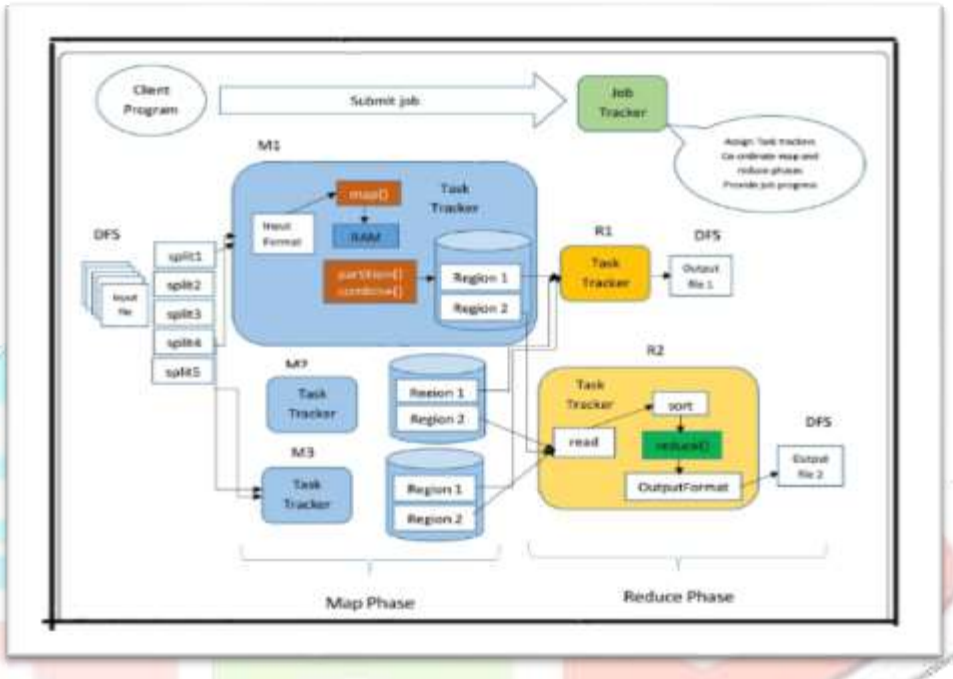

Fig. 9 MapReduce Work Flow of Hadoop [1]

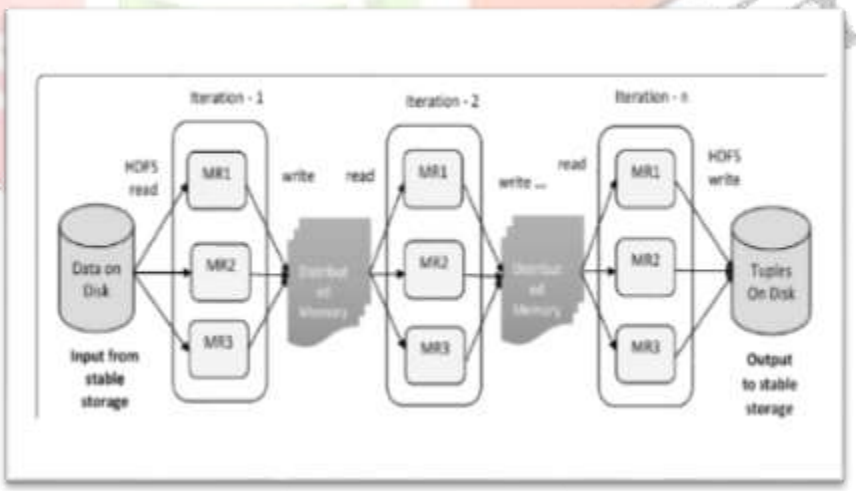

Fig. 10 Spark Work Flow [7]

Sr. No.	Criteria	Apache Hadoop	Apache Spark
1	**Processing Model**	Disk-based processing, which is slower compared to Spark. [1]	In-memory processing for faster data handling.[1]
2	**Data Handling**	More suitable for batch processing.[7]	Efficient for real-time data processing and analytics.[7]
3	**Execution Speed**	Slower as it relies on reading and writing intermediate data to disk	Spark is generally faster (up to 2–14x) for smaller datasets due to in-memory computing. [6]
4	**Ease of Use**	API is more complex, and writing MapReduce jobs can be cumbersome.	Provides simple APIs for Java, Python, Scala, and R, making it easier to write and maintain programs.
5	**Fault Tolerance**	HDFS provides automatic fault tolerance at the storage level. [9]	Relies on RDDs (Resilient Distributed Datasets) for fault tolerance and recovery. [7]
6	**Streaming Support**	Does not support real-time stream processing; Hadoop is batch-oriented.	Supports, real-time stream processing through Spark Streaming**.**
7	**Resource Utilization**	More efficient in terms of memory usage for large datasets but at the cost of performance	Requires more memory and resources, especially for large datasets, which can become a bottleneck
8	**Scalability**	Scales efficiently across thousands of nodes, relying on HDFS to handle large-scale data.	Scales well but requires fine-tuning of configurations like memory allocation to avoid performance degradation
9	**Use Cases**	More suited for large-scale batch processing, such as log analysis and large file processing.	Ideal for machine learning, real-time analytics, and graph processing

Table: Difference Between Apache Hadoop and Apache Spark

X. FUTURE INTEGRATIONS FOR MORE BETTERMENT

Based on our Study of all the referred study we have come to the conclusion that if there is integration of these technologies then

the limitations that Apache Spark and Apache Hadoop are currently facing can be reduced to a greater extend. Through which they can compete with other technologies and frameworks which are working and are industrial competitors for this two Frameworks such as Apache Flume, Google Cloud Dataflow, MongoDB, MySQL, etc. We can't be sure about our claims but can guarantee that these integrations will help these tools to boost up their working in the field of Big Data Analysis, and will certainly improve user experience.

Future Integrations for Apache Hadoop:

So, as we discussed above let's see what are the necessary Future integrations for Apache Hadoop, which are necessary from the perspective of industry and user experience.

Below are the listed integrations for Apache Hadoop:

1. Integration with Cloud:

As we know that Apache Hadoop uses disk storage mechanism which is not enough for today's growth rate and frequent change in data, thus it is necessary that Hadoop's integration with platforms like AWS, Google

Cloud and Microsoft Azure will enhance its memory storage, which allows users to store and process this vast amount of data, which will also improve the scalability of the Hadoop.

2. Support for Real-time Data Processing:

As we discussed in our study that batch processing mechanism for processing the data which is not compatible with today's rapidly growing world. As we know that with the growth of technology there is growth of data also. As initially data was not generated through machines and sensors but it is now generated as well. That's why integrating it with real-time stream processing tools like Apache Flink and Apache Kafka would open up new opportunities for real-time analytics. These integrations could make it easier to process and analyze data in real-time, enabling faster decision-making.

3. Improved Machine Learning Capabilities:

As one of the important tasks of today's world is to train machine learning models which is not possible in Hadoop but is the necessity of today's users. Hence the integration of machine learning platforms like TensorFlow or Apache Spark MLib [6] could be improved to support large-scale training of models. This could allow organizations to run complex machine learning algorithms on massive datasets more efficiently.

4. Better Integration with IoT Devices:

As in the initial stages of Big Data when it was introduced there was no IoT devices came into the pictures thus, developers of Hadoop were unaware that the data can also be produced by sensors though through the years of evolution and improvement it still doesn't integrate with IoT Devices [1]. It can be integrated seamlessly with IoT data platforms using Frameworks such as Apache Nifi, which will help them to process massive amount of data generated by connected devices.

5. Enhanced Security and privacy:

Security is a biggest concern when we talk about data as we know that and studying through this review paper that plays a key role in the growth of any industry. Thus, Hadoop is very less secure as compared to other Big Data Tools and Frameworks even though it uses Kerberos [1] it's still not secure and there are chances of data getting lost or corrupted. Hence it will be more feasible and better to integrate it with better security frameworks such as ApacheRanger, Apache Atlas, or Apache Knox.

6. AI and Deep Learning Integrations:

Hadoop's integration with AI and deep learning technologies will be key for processing unstructured data like images and videos. Frameworks like H2O.ai and DeepLearning4j could help Hadoop users harness the power of deep learning algorithms at scale.

Future Integration for Apache Spark:

1. Advanced Real-time Streaming using Apache Kafka:

As we know that spark stream real-time data using Spark Stream [6] which was mentioned above in the study, but it stills lacks and uses micro-batches [6], which is not ideal for applications which need low-latency event processing. While on the other hand

Apache Kafka, has highly optimized for real-time data streams and message queuing [3] which makes faster data processing as compared to Apache Spark. Hence a deeper integration with Apache Kafka could enable true real-time streaming in Spark, which

eliminate its reliance on micro-batch which eventually reduces data latency.

2. GPU Acceleration for Machine Learning and AI:

Through our study of these referred studie's we got a very good knowledge about Spark MLib [6] which helps in training machine learning models using large datasets which helps advance model training. Though this feature gives an upper edge to Apache Spark as compared to Apache Hadoop [2]. MLib is not so much compatible and lacks to train models in efficientbway and it does not support GPU acceleration which is the need of modern day model training. Hence seamless integration modern Frameworks such as TensorFlow and PyTorch which already support GPU acceleration, which drastically improves speed and efficiency for training complex models. With help of this integrations Spark can become a go-to platform for large scale machine learning and AI tasks.

3. Enhanced Cloud Integration:

As we know that both Hadoop and Spark uses conventional disk storage mechanism for storing the data it does not give enough memory storage space and along with that now a days cloud computing is a dominant trend in big data [1]. Thus, Frameworks that seamlessly integrate with cloud platforms like AWS, Google Cloud, and Microsoft Azure are highly preferred. Spark could further integrate with cloud-native services such as Amazon S3, Google BigQuery, and Azure Data Lake.

4. Support for Advanced Data Formats:

With the growing diversity in data, frameworks that can handle various types of data are advantageous. As we know that there are three basic formats of data Structured, Semi-Structured and Unstructured data [9]. So, throughout the study we got to know that Spark already supports structured data and can store and process this structured data in very efficient manner. Where as tools like Presto and Flink, offer superior flexibility in handling wide range of semi-structured and unstructured data. Hence enhancing spark's support for advanced data formats like AVRO, Parquet, ORC, and JSON will make it more versatile for processing data in various industries.

5. Improved Memory Management with Apache Arrow:

Apache Sparks in memory processing speeds up the computational power and speed but it also leads to high memory consumption and costly infrastructure [7]. Thus, by integrating with the Frameworks such as Apache Arrow, could optimize memory sharing between different data processing frameworks, which improves both performance and resource sharing efficiency. Which can eventually help to handle larger datasets and might be more cost-effective.

6. Native Support for Edge Computing and IoT Data:

As with the rise of Internet of Things (IoT) devices and edge computing there is growth of frequently generated data [1]. Which is necessary to be processed in a faster and efficient. Thus, there are many Frameworks such as Flink and Nifi which offers real time data processing of the data generated from IoT devices. By adding native support for IoT and edge computing workflows, Spark could process data closer to where it's generated, reducing latency and bandwidth costs. This would make Spark an ideal solution for applications like connected vehicles, smart cities, and industrial IoT, where real-time insights are required at the edge.

XI. CONCLUSION

These review paper majorly focuses on the Big Data and the Frameworks used in the Big Data storing, processing and analysis. Through this study we had tried to show case a comparative study of Big Data Frameworks and tools primarily focusing Apache Hadoop and Apache Spark. We tried our best to show case a detailed information about technologies used in both Hadoop and Spark such as **HDFS, MapReduce, Hive, HBase, Sqoop, MLib, RDDs, Graphx,** etc. through this study. Along with this we tried to showcase a comparative study of Apache Hadoop and Apache Spark.

Alon with what are their advantages and limitations compared to each other. Last not but the least we tried our best base on the industrial and the knowledge we gained through the study of the referred studies and articles to showcase what future integrations can be used to improve these technologies to a extend where they can work with real-time growing market. And indulge these frameworks in the world of AI and cloud.

XII. ACKNOWLEDGMENT

We would like to express our heartfelt gratitude to all the authors whose contributions were invaluable for the completion of this review study on Big Data frameworks**: Apache Hadoop and Apache Spark**. Our deepest appreciation goes to **Ahmed, N. Barczak, A.L.C., Susnjak, T. et al.**, whose extensive work laid the foundation for understanding the frameworks and their applications. The research conducted by **Hedayati, S., Maleki, N., and Olsson, T. et al.**, and **S Ibtisum** added significant insights to this study, allowing for a deeper analysis of Big Data processing techniques.

We are also grateful to **Eman Shaikh, Iman Mohiuddin, Yasmeen Alufaisan, and Irum Nahvi** for their valuable contributions in the realm of data processing and framework comparison, which enhanced the scope of this review. The works of **Matei Zaharia, Reynold S. Xin, Patrick Wendell, Tathagata Das, Michael Armbrust, Ankur Dave, Xiangrui Meng, Josh Rosen, Shivram Venkataraman, Michael J. Franklin, T. et al.**, played a pivotal role in furthering my understanding of Spark's core functionalities and performance.

The comprehensive studies **by Salman Salloum, Ruslan Dautov, Xiaojun Chen, Patrick Xiaogang Peng, and Joshua Zhexue Huang** added depth to the comparison of Hadoop and Spark. We also wish to thank **V Srinivas Jonnalagadda, P Srikanth, Krishnamachari Thumati, and Sri Hari Nallamala** for their research contributions, which guided some critical sections of this study.

A special mention goes to **Rahul Beakta**, whose work provided substantial input into the exploration of Hadoop's architecture.

The foundational studies by **Konstantin Shvachko, Hairong Kuang, Sanjay Radia, Robert Chansler, T. White, and J.** Venner contributed immensely to this research, helping to frame the historical development and technical underpinnings of Apache Hadoop.

We extend our sincere gratitude to our guide and the Head of the AI and DS department, **Dr. Manjusha Tatiya**, for her constant

support, insightful feedback, and guidance throughout the course of this study. Her expertise and encouragement made this review possible. Lastly, We are thankful to **Indira College of Engineering and Management** for providing the platform and resources necessary for the completion of this research.

XIII. REFERENCES

[1] **Ahmed, N.Barczak, A.L.C., Susnjak, T.et al. "A Comprehensive Performance analysis of Apache Hadoop and**

Apache Spark for large scale datasets using HiBench". J Big Data 7, December 14 2020.

[2] **Hedayati, S., Maleki, N., Olsson, T. et al. "MapReduce Scheduling Algorithms in Hadoop: A systematic study". J Cloud Comp 12,** October 10 2023.

[3] **S Ibtisum, "A COMPARATIVE STUDY ON DIFFERENT BIG DATA TOOLS",** September 2020

[4] **Eman Shaikh, Iman Mohiuddin, Yasmeen Alufaisan, Irum Nahvi. "Apache Spark: A Big Data Processing Engine"**

November 2019.

[5] **Matei Zaharia, Reynold S. Xin, Patrick Wendell, Tathagata Das, Michael Armbrust, Ankur Dave, Xiangrui Meng,**

Josh Rosen, Shivram Venkataraman, Michael J.Franklin, T .et al. "Apache Spark a Unified Engine for Big Data Processing". October 2016.

[6] **Salman Salloum, Ruslan Dautov, Xiaojun Chen, Patrick Xiaogang Peng, Joshua Zhexue Huang. "Big data analytics on Apache Spark".** July 2016.

[7] **V Srinivas Jonnalagadda, P Srikanth, Krishnamachari Thumati, Sri Hari Nallamala. "A Review Study of Apache Spark in Big Data Processing".** May-June 2016.

[8] **Rahul Beakta, "Big Data and Hadoop",** January 2015

[9] **Jaskaran Singh, Varun Singla, "Big Data: Tools and Technologies in Big Data",** February 15 2015

[10] **Konstantin Shvachko, Hairong Kuang, Sanjay Radia, Robert Chansler, "The Hadoop Distributed File System",** June 28 2010 [Online]

The Hadoop Distributed File System | IEEE Conference Publication |

[11] **T. White,** "Hadoop: The Definitive Guide" in , OReilly

Media, Yahoo! Press, June 2009.[Online]

Hadoop: The Definitive Guide Tom white -Google Book

[12] **J.Venner, "Pro Hadoop",** Apres", June 2009 [Online]

Pro Hadoop - Jason Venner- Google Book