MINAR International Journal of Applied Sciences and Technology
ISSN: 2717-8234
Article type: Research Article

Received: 12/01/2023 **Accepted:** 20/02/2023 **Published:** 01/03/2023

A COMPREHENSIVE STUDY ON BIG DATA FRAMEWORKS

Nagham A. SULTAN[1]

Computer Science Department, College of Computer Science and Mathematics,University of Mosul, Iraq

Dhuha B. ABDULLAH [2]

Computer Science Department, College of Computer Science and Mathematics,University of Mosul, Iraq

Abstract

With the advent of cloud computing technology, the generation of data from various sources has increased during the last few years. The current data processing technology must handle the enormous volumes of newly created data. Therefore, the studies in the literature have concentrated on big data, which has enormous volumes of almost unstructured data. Dealing with such data needs well-designed frameworks that fulfil developers' needs and fit colourful purposes. Moreover, these frameworks can use for storing, processing, structuring, and analyzing data. The main problem facing cloud computing developers is selecting the most suitable framework for their applications. The literature includes many works on these frameworks. However, there is still a severe gap in providing comprehensive studies on this crucial area of research. Hence, this article presents a novel comprehensive comparison among the most popular frameworks for big data, such as Apache Hadoop, Apache Spark, Apache Flink, Apache Storm, and MongoDB. In addition, the main characteristics of each framework in terms of advantages and drawbacks are also deeply investigated in this article. Our research provides a comprehensive analysis of various metrics related to data processing, including data flow, computational model, overall performance, fault tolerance, scalability, interval processing, language support, latency, and processing speed. To our knowledge, no previous research has conducted a detailed study of all these characteristics simultaneously. Therefore, our study contributes significantly to the understanding of the factors that impact data processing and provides valuable insights for practitioners and researchers in the field.

Keywords: Big data; Frameworks; Metrics of Big Data.

http://dx.doi.org/10.47832/2717-8234.14.4

[1] naghamajeel@uomosul.edu.iq

Introduction

Advanced digital artefacts and their uses naturally produce massive data. Examples of contemporary digital technology ingrained in our daily lives include mobile phones, sensors, and social networks resulting in massive amounts of data, also known as big data. However, big data has more properties than just a lot of data. Four opposite attributes are frequently defined Bigdata: volume, velocity, versatility, and accuracy [1]. Big data refers to new methods and tools for storing, exchanging, acquiring, managing, and studying massive data sets with varied architectures. Big data cannot be stored using conventional data management methods since it can be unstructured, semi-structured, or structured. Parallelism utilizes these data effectively and affordably [2-3]. Partitioning, computation, and storage have all seen the introduction of abstract parallel processing models. Most methods begin with partitioning, where a sizable data set is divided across several processing nodes, with each node performing operations on the allocated partitioned data. Although there are some parallel processing variations, it is believed that each processing machine must carry out an identical set of activities in a shared-nothing architecture. Most models transmit their partial output to the master node for production and combine the results to produce the final product [4].

Big data tools and frameworks were developed due to the three V's, which are critical aspects of big data. Big data frameworks are intended to be used in scenarios that traditional systems cannot manage. The following attributes specifically enhance the potential of big data tools and frameworks: First, data distribution comprises breaking the data into smaller blocks and distributing those blocks across the cluster's available nodes. The data is stored in the Distributed File System (DFS) and is prepared for parallel processing following the distribution and fault tolerance. Every block of data is duplicated over many nodes to guarantee that it is always accessible, even if one of the node servers is unavailable for whatever reason—tools and frameworks for big data [5].

1. Related Works

This section will discuss related papers that relied on criteria for classification and the use of big data frameworks:

The authors Shikha Soni et al. presented a paper in [6] discussing big data utilizing frameworks such as Spark, Hadoop, Storm, Flink, and Samza. They also discussed the difficulties they encounter when using big data, including Data Analysis and Storage, Computational Complexities and Scalability, Visualization of Data and Information Security.

Saeed Ullah et al. recommended using Hadoop Spark, Flink, Storm, and Samza in their study [7]. The result addresses choosing possible resource providers for big data applications based on specific criteria. It organizes critical large data resource management systems according to how a cloud computing environment is set up. They examined the advantages and disadvantages of various big data resource management frameworks. They assessed the effectiveness of resource management engines using seven essential criteria: fault tolerance, processing speed, machine learning, scalability, security, low latency, and dataset size.

In their study, Sonia Saini et al. [8] concentrated on the Analysis of Twitter (Social) data using R and MongoDB. They talk about the R data analysis packages that may be used to examine EHR data and tweets. They talked about how MongoDB facilitates the mapping of tweets to documents, offers fundamental operations like aggregation, and what all analysis features/packages R offers for studying the data in the medical area. In [9], Lukman Ab. Rahim et al. employed seismic data and Apache Hadoop's Map Reduce function to boost the system's overall performance for seismic algorithms. The outcome showed that Hadoop (with a default block size of 64 MB) could perform comparably to the original parallel processing

technique but that the performance of the original parallel processing was always superior. They investigated the bottlenecks and enhanced the seismic algorithms system's overall performance. The drawback is that it Does not contain real-time Analysis like the Spark framework.

After reviewing the previous papers, the researchers relied on some criteria in determining the use of big data frameworks. Nevertheless, they should have addressed all the existing measures that determine the use of the appropriate framework. Therefore, more standards will be taken to give an additional aspect and better credibility when using the Big Data framework.

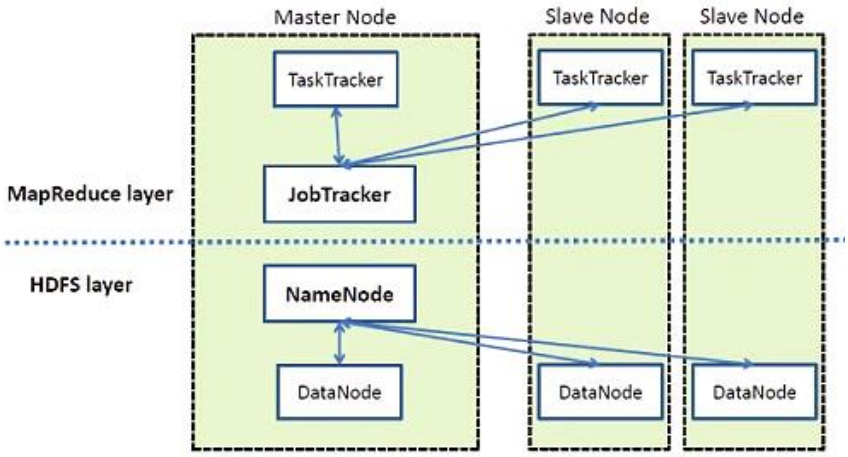
2. Big Data Frameworks

In this section, big data frameworks will be introduced in some detail as follows:

2.1. Hadoop

Doug Cutting, a Yahoo employee, and Mike Cafarella, a professor at the University of Michigan, created the framework in its initial form. Later, it underwent several years of evolution to arrive at its current stable version. Since its first release, Hadoop has drawn increased scientific interest and has been embraced by several major corporations, including Yahoo, Amazon, Facebook, eBay, and Adobe, which has sped up the framework's evolution [10].

The Hadoop Software Library, a component of the Apache Projects, is a framework that enables the distributed processing of large amounts of data using a small number of machines. Map Reduce and the Hadoop Distributed File System (HDFS) are combined to form Hadoop. While HDFS is responsible for storing the data in a file system, Map Reduce is accountable for processing the data [11]. MapReduce, Hadoop Kernel, Hadoop Distributed File System (HDFS), and other diverse components, including Base, Apache Hive, and Zookeeper, are all part of the current Apache Hadoop framework, which is depicted in Figure 1, [2].



The open-source software framework Apache Hadoop typically implements MapReduce for distributed storage and processing across massive datasets on computer clusters. MapReduce is the processing component of Apache Hadoop, which is made up of a storage component based on the Hadoop Distributed File System (HDFS). As a result, MapReduce can be used in various large-scale computations that are tolerant of hardware issues. The MapReduce calculation process is depicted in Figure 2 [12].

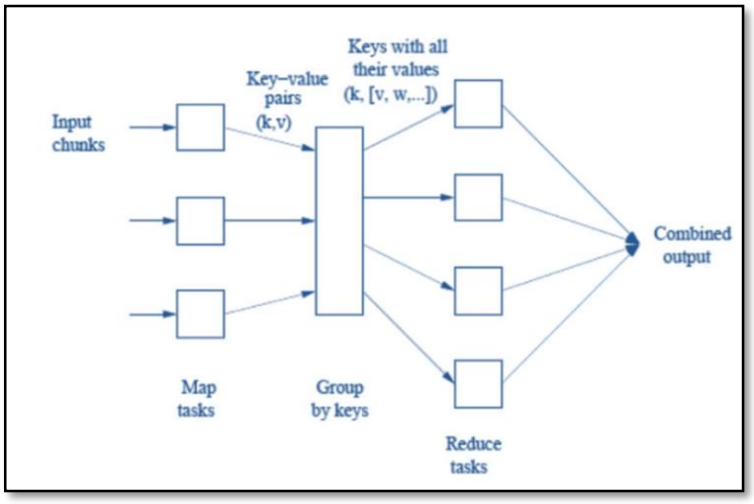


Figure 2: MapReduce computation

2.2. Spark

Spark is an open-source distributed general-purpose cluster computing platform for processing massive amounts of data. Spark is one of the go-to technologies for developers and data scientists interested in big data since it is the open-source framework for distributed and parallel processing currently being developed the most actively [13]. Spark Streaming expands the Spark core API. It makes the development of fault-tolerant real-time data stream processing straightforward. Only Spark's robust Spark Core API and other libraries can handle big data analytics and Machine Learning. The Spark ecosystem (Illustrated in Figure 3) includes the Spark framework's core engine as well as the following components: Spark SQL [Data Frames and Data Sets], Spark Streaming [Near and Real-Time data processing], ML, Graph Analytics [GraphX] data processing, Spark Cassandra Connector, and Spark R Integration [14].

Spark Streaming means real-time streaming data from sources like Twitter, Flume, Kafka, Kinesis, ZeroMQ, or TCP connections are processed. Real-time processing is done using DStream, and the supporting frameworks: Spark Streaming, Storm, and Apache Samza. However, Spark SQL-Relational DB Processing meant: A structured data query that uses a distributed dataset, such as a Spark data frame or schema RDD, referred to as "Spark SQL." SQL and HiveQL queries are running. Some features include database connectivity (ODBC/JDBC), unified data access (query sources—Hive tables, JSON), and integrated RDD with Spark SQL queries.

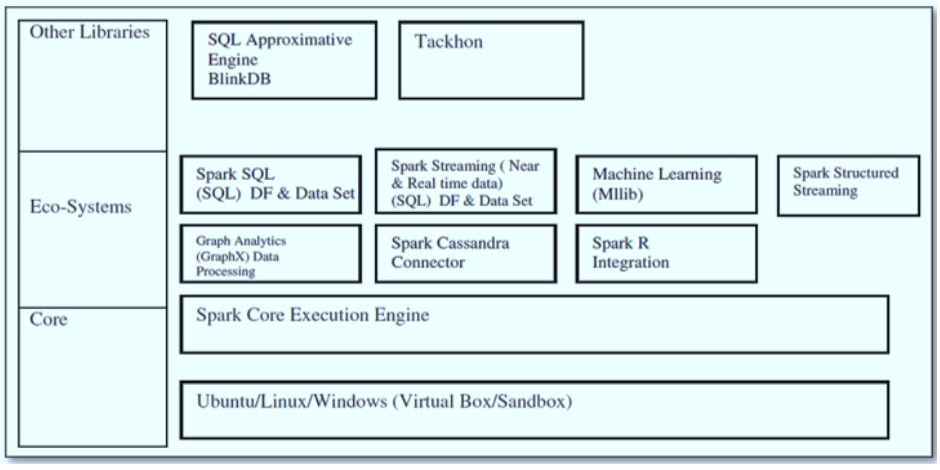


Figure 3: Apache Spark ecosystem

A machine learning library known as MLlib is responsible for the production of both standard learning algorithms and tools. This is what was intended by the term "Spark MLlib-Library of Machine Learning Algorithms." It is made up of a variety of approach sets and the algorithms that go along with them. Spark The term "GraphX," which stands for "Graph Analytics/Visual Analytics," refers to the fact that the graph is made up of edges and vertices. Node is another word for the vertex, and it can refer to either a location or an individual who has connected relationships that are characterized by edges. Suppose that the entire Facebook site is a social graph; it would be very difficult to figure out who is connected to whom because there would be so much information to process. With Resilient Distributed Datasets (RDD), which in Apache Spark are merely vertices, Directed Acyclic Graphs (DAG) can be constructed; updates are only used to build edges. When traveling from one location to another, there are many distinct routes that one can take. Graph processing allows for the efficient generation of a shortest path between two sites. This is made possible if we represent locations as vertices and highways as edges[15], [16].

2.3. Flink

Flink offers both real-time and batch data types with high-throughput and low-latency output. When a machine fails, Flink is highly fault-tolerant. Numerous programming languages, including Java, Scala, Python, and SQL, are supported by Flink Programs. In addition to using particular data structures and algorithms for the batch versions of operations like join and grouping, Flink also includes a dedicated API for processing static data sets. The result is that, on top of a streaming runtime, Flink presents itself as a complete and adequate batch processor [17]. Its benefits include consistent and dependable performance, quick data processing, user-friendly APIs, and an open-source community that encourages development and support from all developers [18]. Its data analysis is expressive, declarative, quick, and effective for both batch and real-time data. The Distributed Data-Driven Engines' Complexity is reduced via Apache Flink [19]. Flink is a program that has been optimized; using an optimization method in the built-in application interface produces results. Due to the processing of the altered portion, it is quicker than the Spark.

In contrast to Spark and Hadoop, Flink returns the result with higher latency and throughput. It is a platform that only supports streaming. It has its platform for managing memory and does not rely on Java's garbage collection [20]. Figure 4 displays how Flink is used in batch and stream data processing. It combines MapReduce's scalability with its programming flexibility. For cluster environments, it is appropriate. The speed advantage over Hadoop-MapReduce is 100 times. By incorporating conventional database ideas, Flink provides a lower level of complexity. Additionally, it provides a very high degree of scalability [19].

The Flink architecture is a distributed system that needs resources to be allocated and managed to run the streaming application. This integrates with famous clusters like Hadoop YARN and the Apache Mesons resource manager and may also be used as a standalone cluster. Two processes make up Flink: Job Manager and Task Manager. The Task Managers are the workers who execute the components as parallel programs, while the Job Manager coordinates the Flink System. The same JVM is utilized for the Single Job and Task Manager when the system is started in local mode. When a program is uploaded, it preprocesses it and converts it into parallel data that the Job Manager and Task Manager can use to run [17], [19]- [20].

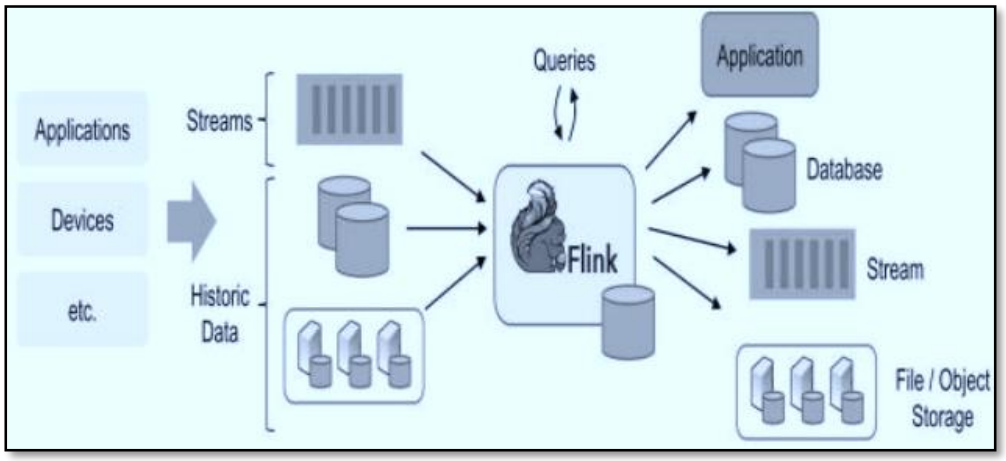


Figure 4: Usage of Apache Flink

2.4. Storm

Apache Storm has been used to implement the real-time data processing topology. This system is a part of the Hadoopecosystem [21]. It is distributed real-time computing framework that is free and open-source, and it offers an interface for building event-based elaborations across a group of physical nodes. By utilizing the topology abstraction, users can put the queries to be computed into practice. The Storm Cluster, which is in charge of deploying and running, will receive such a topology [22]. It offers complete guarantees of successful message processing, is fault-tolerant, and is horizontally scalable. The application logic for the Storm can be written in any language, regardless of the programming language [23]. The fault-tolerant approach to handling failures offers the required level of reliability. At the same time, the distributed computations architecture ensures high performance and a wide range of flexibility in creating spouts that support various data streams. Support for various technologies is one of Storm's key advantages. The adaptable Apache Storm platform supports different technologies (first of all, Java, and also .NET, Python, Etc.). It also permits a combination of them [21].

Storm offers a solid foundation for work; you may afterwards add and delete processing nodes from its cluster. Additionally, it offers an API for keeping track of the system's overall health and permits applications to operate on top of it to report performance data. Storm efficiently handles unrestricted data streams. It comprises a platform that controls and runs user-written application code. The stream, an infinite series of tuples, is a crucial idea in Storm (data items). Developers can create processing applications using the computing power of every computer in a cluster using the framework provided by Storm software. The platform should be able to automatically expand and contract as necessary due to the diverse processing requirements of such applications. Unfortunately, this capability is unavailable on the current Storm platform [24]. The Storm is appropriate for sequential and iterative processing, machine learning, and real-time data analysis. A directed acyclic graph (DAG) describes a storm topology. The program DAG's edges stand in for data transport. Spouts and bolts are the categories into which the DAG's nodes are classified [25].

2.5. MongoDB

The most well-known NoSQL database is MongoDB. It is an excellent tool for creating data warehouses, especially given its capacity to exploit so-called "sharding-nothing cluster architecture fully." Since it is open-source, it is perfect for creating high-performance data warehouses. Binary encoded JavaScript Object Notation (JSON), called BSON data

structures, are supported by MongoDB's features for storing complicated data types. Enables developers to use or store Java Script, functions, and values on the server side; supports a simple-to-use protocol to store large files and file metadata; and will provide fast serial performance for a single client and use memory-mapped files for fast performance. Large binary files like images and videos are distributed and stored using this protocol. This feature of MongoDB has led to many projects considering utilizing MongoDB rather than a relational database [2].

MongoDB has document-oriented storage; the data is kept in documents that resemble JSON. Any character may be used to index it. MongoDB is employed in big data, Data Hub, Content Management and Delivery, Mobile and Social Infrastructure, and User Data Management. High performance is one of the main features of MongoDB; it Provides high-performance data continuity. In particular, it allows for embedded data models, which reduce the I/O activity of the database system. Indexes also support faster queries and can use embedded documents and array keys in their keys; plus, they have a rich query language that allows reading, write, and delete operations, data aggregation, and text search. It also has high availability, i.e., the MongoDB replica set replication capability provides data redundancy and automatic failover. A replica set is a group of MongoDB servers that all maintain the same data set, enhancing data availability and ensuring redundancy. As for horizontal scalability, one of MongoDB's main features, data is split between a set of machines by hashing [26].

3. FRAMEWORKS COMPARISON

The set of criteria that we will address in this study is (Data Processing, Streaming Engine, data flow, computation model, task performance, fault tolerance, scalability, Iterative Processing, Language Support, latency, and Processing Speed) and as follows:

- **Data Processing [2], [17], [27]**

✓ Apache Hadoop: Apache Hadoop was created for batch processing. It simultaneously processes a large amount of data as input and outputs the results. Batch processing of vast amounts of data is quite efficient. An output is delayed because of the volume of data and the system's computing power.

✓ Apache Spark: Apache Spark is another part of the Hadoop Ecosystem. It is primarily a batch processing system, albeit it also provides stream processing.

✓ Apache Flink: enables batch and streaming processing in a single runtime.

✓ Apache storm: A distributed system for processing massive data in real-time is called Apache Storm. Despite being stateless, Storm uses Apache Zookeeper to handle the distributed environment and cluster state. It is easy to use and allows for the concurrent execution of various transformations on real-time data.

✓ MongoDB: The best database for developing applications is MongoDB. Because of its adaptable data schema and inherent scalability as a NoSQL database, developers favor this database. Development teams can iterate and pivot rapidly and effectively because of these characteristics.

- **Streaming Engine [2], [25], [28], [29]**

✓ Apache Hadoop: The batch-oriented processing tool Map-reduce. It receives a sizable data set as input, processes it all at once, and outputs the outcome.

✓ Apache Spark: All workloads in Apache Spark are executed in micro-batches. However, it falls short in use cases when we must handle huge amounts of real-time data and deliver conclusions immediately.

✓ Apache Flink: For all workloads, including batch, micro-batch, streaming, and SQL, Apache Flink leverages streams. A batch is a small collection of streaming data.

✓ Apache Strom: This framework for streaming data has the highest ingestion rates. Storm moves the data to the code, as opposed to Hadoop, which moves the code to the data. This behavior makes more sense in a stream-processing system since, unlike in a batch job, the data set isn't known in advance. Additionally, the code is constantly being passed through the dataset.

✓ MongoDB: Many applications depend on streaming data in one way or another. Over time, MongoDB has improved, providing new features and functionalities to handle these workloads. With just a few lines of code, you can easily stream data to and from MongoDB using the MongoDB Spark Connector version 10.0.

- **Data Flow [10], [11], [17]**

✓ Apache Hadoop: There are no loops in the MapReduce processing data flow. There is a progression to it. You advance through each step by using an output from the stage before and creating input for the stage after.

✓ Apache Spark: Spark represents the Machine Learning algorithm as a (DAG) direct acyclic graph, even though it is a cyclic data flow.

✓ Apache Flink: Flink approaches things differently than other apps. In run time, it supports the controlled cyclic dependency graph. This makes it easier to efficiently represent machine learning algorithms.

✓ Apache Storm The directed acyclic graph (DAG) topology of Storm uses spouts, bolts, and streams to process data.

✓ MongoDB: It is up to data experts to decide whether to incorporate the information or acquire it independently in a set of documents when building data models in MongoDB. Thus, the two most useful MongoDB data modelling principles are embedded and unified data models.

- **Computation Model [15], [27], [30]**

✓ Apache Hadoop: The batch-oriented model was adopted by MapReduce in Apache Hadoop. The batch is handling data that is at rest. It simultaneously takes in a lot of input, processes it, and then outputs the results.

✓ Apache Spark: The micro-batch mode is the foundation of Apache Spark.

Apache Flink: Flink uses an operator-based streaming approach with continuous flow. A continuous flow operator does not wait to gather or process data; instead, they do so as it comes in. Meant Flink is built on the computational model with operators.

✓ Apache Storm: is a distributed real-time computing system that is free and open source.

✓ MongoDB: An application frequently has to extract a value from source data that is kept in a database. When dealing with huge data sets or situations where numerous papers need to be scanned, computing a new value could demand a significant amount of CPU power. It might be preferable to save a computed value to the database at the outset if that value is frequently requested. As a result, only one read operation is necessary when the program requests data.

- **Performance [11], [23], [31]**

✓ Apache Hadoop: Only batch processing is supported by Apache Hadoop. As a result of not processing streamed data, it performs less quickly than Spark and Flink.

✓ Apache Spark: Despite having a solid community history, Apache Spark is today

regarded as having the most established one. However, its stream processing is less efficient than Apache Flink due to the utilization of micro-batch processing. Performance is significantly improved because memory data access happens in nanoseconds instead of milliseconds on a disk drive.

✓ Apache Flink: About other data processing systems, Apache Flink performs remarkably well. Apache Flink uses native closed-loop iteration operators to speed up machine learning and graph processing when comparing Hadoop, Spark. Apache Flink performs exceptionally well overall when measured against other data processing platforms.

✓ Apache Storm: optimizing Apache Storm's speed is a vital but time-consuming task because hundreds of parameters can be adjusted and roughly a thousand different configurations may be used.

✓ MongoDB: You might need to evaluate the performance of the application and its database as you create and manage apps using MongoDB. The number of open database connections, hardware accessibility, and database access mechanisms frequently factor in performance degradation. High-performance data persistence is offered by MongoDB. Support for embedded data models, in particular, lowers the amount of I/O activity on database systems, while indexes provide quicker queries and can incorporate keys from embedded documents and arrays.

- **Fault tolerance [11], [17], [30]**

✓ Apache Hadoop: MapReduce is very fault-tolerant in Apache Hadoop. In the event of any Hadoop failure, the application does not need to be restarted from scratch.

✓ Apache Spark: Without additional code or configuration, Spark Streaming provides exactly-once semantics out of the box and recovers lost work.

✓ Apache Flink: Chandy-Lamport distributed snapshots serve as the foundation for Apache Flink's fault tolerance system. Because of the mechanism's small weight, great throughput rates are maintained while also offering solid consistency guarantees.

✓ Apache Storm: Based on the "fail fast, auto restart" strategy, Apache Storm is fault-tolerant and handles mistakes very effectively. This enables it to restart the process whenever a node fails without disrupting the overall operation. The Storm is a fault-tolerant engine as a result of this characteristic. If one or more nodes fail or a message is lost, it ensures that each tuple will be processed "at least once or exactly once."

✓ MongoDB: Deploying numerous copies of your database to other servers or model instances is known as replication. It is referred to as a replica set in MongoDB. Because of this, Mongo]DB offers fault tolerance.

- **Scalability [6], [32], [33]**

✓ Apache Hadoop: Tens of thousands of Nodes have been used in production using MapReduce, which has great scalability potential.

✓ Apache Spark: The cluster can continue to grow by n nodes because it is very scalable. A known huge spark cluster contains 8000 nodes.

✓ Apache Flink: The Apache Flink cluster can continue to grow by n nodes because it is very scalable. An enormous Flink cluster is one with many thousands of nodes.

✓ Apache Storm: it is a highly scalable and quick framework that makes it simple to distribute work across several threads, JVMs, or machines—all without having to modify your code to scale in that way. With fewer than 20 nodes, it can already handle a

million jobs per second for its consumers. Storm can maintain performance even while the workload is rising by linearly adding resources. It can scale up very well.

✓ MongoDB: Because its data is not relationally linked, MongoDB is scalable. The information is self-contained and is kept in documents that resemble JSON. Horizontal scaling makes it simple to spread these documents over numerous nodes. MongoDB allows you to scale your collections horizontally by hashing the data into smaller segments or vertically by adding more resources to the collection.

- **Iterative Processing [1], [6], [27], [33]**

✓ Apache Hadoop: Iterative processing is not supported by Apache Hadoop.

Apache Spark: This program batches the data iterations. Each iteration in Spark needs to be scheduled and carried out independently. It is founded on non-native iteration, which is carried out externally like standard for-loops.

✓ Apache Flink: This program uses its streaming design to iterate data. The job's efficiency can be greatly improved by instructing it to just process the portions of the data that have truly changed. Iterate and Delta Iterate are two separate iteration operations offered by the Flink API.

✓ Apache Storm: it is a real-time streaming solution because its processing architecture operates directly on tuple streams, one record at a time. Micro-batches are a new option made available by the Trident API.

✓ MongoDB: When compared to MongoDB, Hadoop's iterative Map-Reduce procedure for big data is particularly slow because iterative tasks necessitate numerous Map and Reduce operations to be completed. This procedure multiplies the files between the map and reduces the jobs, rendering the map utterly useless for sophisticated analysis. MongoDB's most popular and extensively used database established the assembly pipeline structure to make up for this setback.

- **Language Support [2], [17], [34], [35]**

✓ Apache Hadoop: It mostly supports Java, although it also supports Ruby, C, C++, Groovy, Python, and Perl.

✓ Apache Spark: It supports R, Python, Scala, and Java. Scala is used to implementing Spark. It offers Java, Python, and R API, among other languages.

✓ Apache Flink: It supports R, Python, Scala, and Java. Java is used to implement Flink. It also offers Scala API.

✓ Apache Storm: Storm supports almost every programming language thanks to its multi-language functionality. The languages Java, Clojure, and Scala are all compatible with the Trident API for streaming and processing.

✓ MongoDB: C, C++, C#, Go, Java, Node.js, PHP, Python, Ruby, Rust, Scala, and Swift are the languages that are officially supported and compatible with MongoDB.

- **Latency [24], [33], [36]**

✓ Apache Hadoop: Due to its support for various data types, structures, and volumes, Hadoop's MapReduce framework is comparatively slower than other frameworks. Hadoophas a higher latency than Spark and Flink because of this.

Apache Spark: Although Apache Spark is a batch processing system, it is significantly faster than Hadoop MapReduce because it uses RDDs to store a considerable percentage of the input data in memory and stores intermediate data there before writing it to disk when it is finished processing it or as needed.

✓ Apache Spark has a higher latency when compared to Apache Flink.

✓ Apache Flink: With little configuration work, Apache Flink's data streaming runtime provides high throughput and low latency.

✓ Apache Storm: It offers greater low latency with fewer constraints, and its latency is measured in milliseconds.

✓ MongoDB: MongoDB typically emerges victorious. Thanks to master replication and redundancy, MongoDB can take massive amounts of unstructured data significantly faster than MySQL. You might gain a lot from this feature depending on the types of data you gather.

- **Processing Speed [6], [15], [27], [33]**

✓ Apache Hadoop: MapReduce operations in Apache Hadoop are slower than Spark and Flink. The MapReduce-based execution causes the delay because it generates a lot of intermediate data and exchanges a lot of data across nodes, resulting in a significant increase in disk IO latency. Furthermore, to facilitate job recovery from errors, it must persist a large amount of data on a disk for synchronization across phases. Additionally, there are no methods in MapReduce for caching all data subsets in memory.

✓ Apache Spark: Because it uses RDD to cache a significant chunk of the input data in memory and stores intermediate data there before publishing it to disk when done or as needed, Apache Spark executes operations more quickly than MapReduce. How much better Spark is than Hadoop MapReduce is shown by the fact that it is 100 times faster than MapReduce. Spark's processing model is slower than Flink's.

✓ Apache Flink: It completes processes faster than Spark due to its streaming nature. Task performance is enhanced by instructing Flink to just process the data portions that have changed. Using Flink, data processing happens quite quickly.

✓ Apache Storm: the capacity to do many calculations at the same speed under increased load. According to Fast benchmarking, each node can handle one million 100-byte messages per second.

✓ MongoDB: Thanks to in-memory algorithms, it can quickly analyze massive amounts of real-time data. A NoSQL database is MongoDB. Its structure is flexible. MongoDB is a fantastic option for storing, querying, and analyzing big data since it holds enormous amounts of data in a naturally traversable design.

After summarizing all of the above, we can conclude Table 1 as follows:

Table 1: Comparison Between the Five Frameworks

FACTOR	HADOOP	SPARK	FLINK	STORM	MongoDB
Data Processing	Batch only	Hybrid	Hybrid	Streaming only	Real-time
Streaming Engine	No	Yes	Yes	Yes	Yes
Data Flow	No loops in the Map Reduce processing data flow	Direct acyclic graph (DAG)	controlled cyclic graph	Direct acyclic graph (DAG)	Embedded data and Unified data model
Computation Model	In Disk	In Memory	In Memory	In Memory	In Memory
Performance	Good	Very Good	Excellent	Very Good	Very Good
Fault Tolerance	Yes	Yes	Yes	Yes	Yes
Scalability	Yes	Yes	Yes	Yes	Yes
Iterative Processing	No	Yes	Yes	Yes	Yes
Language Support	C, C++, Ruby, Groovy, Perl, and Python.	R, Python, Scala, and Java	R, Python, Scala, and Java	Java, Clojure, and Scala	C, C++, C#, Go, Java, Node.js, PHP, Python, Ruby, Rust, Scala
Latency	Higher	Medium	Low	Low	Low
Processing Speed	Slower than Spark and Flink	Faster than Hadoop and slower than Flink.	More quickly than others	Fast	Fast

We can see from the table that Hadoop is an excellent choice to implement at a lower cost compared to other frameworks when working with batch workloads and is not time-sensitive. As for streaming workloads, Storm can provide processing with very low latency, and when we have a hybrid workload, Spark provides high-speed batch processing and micro-batch processing for streaming data. In contrast, Fink provides accurate stream processing with batch processing support with low latency. All of these frameworks run a fault tolerance; when a node dies, the worker will be restarted on another node automatically. To achieve fault tolerance, these frameworks use strategies such as data and computation replication, as well as task rescheduling. Data replication involves copying data onto multiple nodes within the cluster to ensure that it can be recovered from the other nodes in case of an outage, thereby avoiding any data loss. Computation replication ensures that any computations that were running on a failed node are replicated on another node within the cluster to prevent processing interruption and loss of progress. These frameworks are scalable parallel calculations that run across a cluster of machines. In terms of speed, Spark, Flink, Storm, and MongoDB are faster than Hadoop because they store data in

memory, while Hadoop stores data in the hard disk; Spark is slower than Flink in terms of processing latency because it is a micro-batch, so itis slower than Flink and that is having higher latency, Storm is also faster than Spark, as Storm can process over million records per second on a cluster of modest size. In terms of ease of use, they are considered easier than Hadoop because they are complex and challenging. In terms of cost, Spark, Flink, and Storm are more expensive than Hadoop. Spark internally takes advantage of micro-batch for processing, but spark stream has a latency of less than a second; Flink is mainly based on true event streaming, even batch is a particular case of streaming, but Flink internally will process data in the form of a stream, that is why Spark has a higher throughput it because it is working in a micro-batch, it is processing multiple events together, while Flink it has lower throughput. Flink is better than Spark, and Hadoop and Storm keep performance even during increased workloads by adding resources linearly.

4. Conclusion

The main objective of this paper is to study and compare big data frameworks with their features in manipulating data, then produce a good comparison between frameworks by going into deep studding for their abilities from multiple points of view such as data streaming, fault tolerance, data processing, latency, and others features. So, any researcher who tries to enter and work with this field of information must spend a huge amount of time to differentiate between various abilities of big data frameworks, Therefore, this paper will play a role in choosing the appropriate framework. Following a brief introduction and comparison of Hadoop, Flink, Storm, Spark and MongoDB, it is determined that Flink is from a researcher's point of view is the fastest and most reliable framework when compared to other frameworks because it can process data in both streaming and batch form for the same version and is faster and more dependable due to fault tolerance, low latency, and high throughput. It is possible to select a framework for large data that works with parallel processing if you so like. In point of fact, the majority of big data frameworks are designed to support parallel processing and distributed computing in order to manage the large volumes of data that are typically involved in big data applications. This is done for the purpose of handling the workload that is presented by big data. Apache Hadoop, Apache Spark, Apache Flink, and Apache Storm are only few of the popular big data frameworks that offer parallel processing. These frameworks offer distributed computing capabilities, which make it possible to analyze big datasets over a cluster of computers in a parallel manner that is also fault-tolerant.

References

[1] Osman, A.M.S., A novel Big data analytics framework for smart cities. Future Generation Computer Systems, 2019. 91: p. 620-633.

[2] Kumar, P., et al., Analysis and comparative exploration of elastic search, MongoDB and Hadoop big data processing, in Soft Computing: Theories and applications. 2018, Springer. p. 605-615.

[3] Hadeed, W.W., and D.B. Abdullah, Real-Time Based Big data and E-Learning: A Survey and Open Research Issues. AL-Rafidain Journal of Computer Sciences and Mathematics, 2021. 15(2): p. 225-243.

[4] Ordonez, C., S.T. Al-Amin, and X. Zhou. A simple low-cost parallel architecture for big data analytics. in 2020 IEEE International Conference on Big data (Big data). 2020: IEEE.

[5] Buyya, Rajkumar, et al. Big Data: Principles and Paradigms. 1st ed., Morgan Kaufmann, 2016.

[6] Soni, S., J. Singla, and M.M. Yousuf, BIG DATA: FRAMEWORKS AND CHALLENGES. 2018.

[7] Ullah, S., M.D. Awan, and M. Sikander Hayat Khiyal, Big data in cloud computing: A resource management perspective. Scientific Programming, 2018.

[8] Saini, S. and S. Kohli, Healthcare Data Analysis Using R and MongoDB, in Big Data Analytics. 2018, Springer. p. 709-715.

[9] Rahim, L.A., K.M. Kudiri, and S. Bhattacharjee, Framework for parallelization on big data. PloS one, 2019. 14(5): p. e0214044.

[10] Ramírez-Gallego, S., et al., An information theory-based feature selection framework for big data under Apache spark. IEEE Transactions on Systems, Man, and Cybernetics: Systems, 2017. 48(9): p. 1441-1453.

[11] MAMATHA, S., G. NARENDRA, and C.V.V. REDDY, Big data Analysis in Hadoop and MongoDB. 2018.

[12] Tsai, C.-F., W.-C. Lin, and S.-W. Ke, Big data mining with parallel computing: A comparison of distributed and MapReduce methodologies. Journal of Systems and Software, 2016. 122: p. 83-92.

[13] Balqees Agha, Improving Parallel Schedulers in Heterogeneous Distributed Computing for Containerized Big Data,2022, Ph.D. Thesis, College of Computer Science and Mathematics, University of Mosul, Iraq

[14] Venkatesh Saravanakumar, M. and S.M. Hanifa, Processing Using Spark—A Potent of BD Technology, in Big Data Processing Using Spark in Cloud. 2019, Springer. p. 195-215.

[15] Buyya, R., R.N. Calheiros, and A.V. Dastjerdi, Big data: principles and paradigms. 2016: Morgan Kaufmann.

[16] Hamstra, M. and M. Zaharia, Learning Spark: lightning-fast big data analytics. 2013: O'Reilly & Associates.

[17] Thakur, B.S. and K.L. Bansal, Performance Evaluation of Apache Hadoop, Apache Spark, And Apache Flink. Advances in Management, Social Sciences and Technology: p. 93.

[18] Morcos, M., B. Lyu, and S. Kalathur. Solving the 2021 DEBS grand challenge using Apache Flink. in Proceedings of the 15th ACM International Conference on Distributed and Event-based Systems. 2021.

[19] Davidson, G.P., and D.D. Ravindran, Technical Review of Apache Flink for Big data. Int. J. of Aquatic Science, 2021. 12(2): p. 3340-3346.

[20] Yin, F. and F. Shi, A Comparative Survey of Big Data Computing and HPC: From a Parallel Programming Model to a Cluster Architecture. International Journal of Parallel

Programming, 2022. 50(1): p. 27-64.

[21] Batyuk, A. and V. Voityshyn. Apache storm based on topology for real-time processing of streaming data from social networks. in 2016 IEEE First International Conference on Data Stream Mining & Processing (DSMP). 2016: IEEE.

[22] Yan, L., Z. Shuai, and C. Bo. Multi-sensor data fusion system based on Apache storm. in 2017 3rd IEEE International Conference on Computer and Communications (ICCC). 2017: IEEE.

[23] Karunaratne, P., S. Karunasekera, and A. Harwood, distributed stream clustering using micro-clusters on Apache Storm. Journal of Parallel and Distributed Computing, 2017. 108: p. 74-84.

[24] Van Der Veen, J.S., et al. Dynamically scaling Apache storm for the analysis of streaming data. in 2015 IEEE First International Conference on Big Data Computing Service and Applications. 2015: IEEE.

[25] Inoubli, W., et al., An experimental survey on big data frameworks. Future Generation Computer Systems, 2018. 86: p. 546-564.

[26] Agbo, C.C., Q.H. Mahmoud, and J.M. Eklund. A scalable patient monitoring system using Apache Storm. in 2018 IEEE Canadian Conference on Electrical & Computer Engineering (CCECE). 2018: IEEE.

[27] Chen, M., S. Mao, and Y. Liu, Big data: A survey. Mobile networks and applications, 2014. 19(2): p. 171-209.

[28] Carbone, P., et al., Apache Flink: Stream and batch processing in a single engine. Bulletin of the IEEE Computer Society Technical Committee on Data Engineering, 2015. 36(4).

[29] Salloum, S., et al., big data analytics on Apache Spark. International Journal of Data Science and Analytics, 2016. 1(3): p. 145-164.

[30] Jiang, W. and J. Luo, Big data for traffic estimation and prediction: a survey of data and tools. Applied System Innovation, 2022. 5(1): p. 23.

[31] Singh, A., M. Mittal, and N. Kapoor, Data processing framework using Apache and spark technologies in big data, in big data Processing Using Spark in Cloud. 2019, Springer. p. 107-122

[32] Mittal, M., et al., big data processing using Spark in the cloud. 2019: Springer.

[33] Edward, S.G. and N. Sabharwal, Practical MongoDB: Architecting, Developing, and Administering MongoDB. 2015.

[34] Khalid, M. and M.M. Yousaf, A Comparative Analysis of Big Data Frameworks: An Adoption Perspective. Applied Sciences, 2021. 11(22): p. 11033.

[35] Khan, S. and V. Mane, SQL support over MongoDB using metadata. International Journal of Scientific and Research Publications, 2013. 3(10): p. 1-5.

[36] Jaiswal, A., V.K. Dwivedi, and O.P. Yadav. Big data and its Analyzing Tools: A Perspective. in 2020 6th International Conference on Advanced Computing and Communication Systems (ICACCS). 2020: IEEE.