WORLD COMET RESEARCH GROUP

*International Journal of Applied Data Science & Modern Computing*
*Vol. 4, No. 1, 2021*
*Doi: 10.67228/30713498/IJADSMC-2021PI0B9N*
*PP. 01-16*

***Original Article***

# Big Data Processing Using Apache Spark: A Performance Study

**Dr. R. Kartikeyan**

*Assosciate Professor, Anna University, Chennai, India.*

Received: 12-05-2020 Revised: 12-25-2020 Accepted: 12-31-2020 Published: 01-02-2021

***ABSTRACT***

*The rapid growth of big data across domains such as finance, healthcare, and IoT has exposed the limitations of traditional disk-based systems like Hadoop MapReduce, which suffer from high I/O overhead and inflexible batch processing. Apache Spark addresses these challenges with its in-memory, distributed computing model, enabling faster and more efficient data processing. This paper analyzes Spark's performance in large-scale data processing, focusing on computational efficiency, scalability, memory usage, and fault tolerance. A comparative study with Hadoop MapReduce shows that Spark achieves 3× to 20× speed improvements, especially for iterative and in-memory workloads. Experimental results also highlight the impact of memory configuration, data partitioning, and data locality on performance. Despite challenges like memory pressure and resource management, Spark proves to be a powerful solution for optimizing big data analytics pipelines.*

***KEYWORDS***

*Big Data, Apache Spark, Distributed Computing, In-Memory Processing, Hadoop MapReduce, Performance Evaluation, Scalability, Data Analytics, Cluster Computing, Resource Management.*

Copyright © WCRG The Author(s) This article is distributed under a
Creative Commons Attribution-Share A like 4.0 International License.

ISSN: 3071-3498
PP: 01-16.

# *1.INTRODUCTION*

## 1.1. Background

The digital data are increased exponentially, and the paradigm of big data has been created, which is usually defined by the so-called 5Vs: Volume, Velocity, Variety, Veracity, and Value. Classic centralized databases and single node processing models are becoming insufficient to process modern scale, speed and complexity data workloads. Consequently, distributed computing systems have emerged as an essential infrastructure underlying scalable data analytics, whereby computing can be done in parallel across clusters of commodity machines to deliver scalability, fault tolerance and high throughput. One of the most popular and initial frameworks to be used in distributed large-scale processing of big data came in the form of Hadoop MapReduce, introducing a powerful framework of scalable and fault-tolerant batch computing. Nevertheless, it uses a high latency discrete and inflexible execution model based on disk-based information storage as an intermediate, and does not scale well to typical iterative algorithms and interactive analytics workloads. These constraints are further intensified by the fact that applications are requiring low-latency processing and repeated data access even more. To address these problems, Apache Spark has been created as in-memory distributed computing platform that can enable effective data reuse during numerous steps of computation. Spark runs its code more quickly, through abstractions including Resilient Distributed Dataset (RDDs), and higher-level APIs, including DataFrames and Datasets, to avoid disk reads, and convert execution plans into efficient applications. This paradigm change of architecture towards in-memory processing and DAG-based evaluation has made Apache Spark one of the leading models to handle the current big data analytics, and would thus serve effectively in machine learning, interactive queries, and real-time data processing solutions.

## 1.2. Importance of Big Data Processing Using Apache Spark

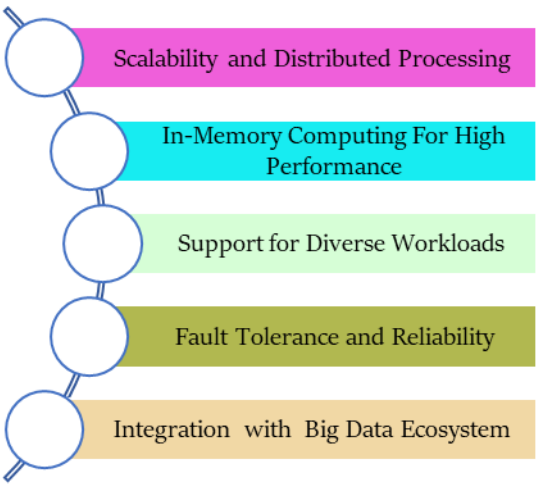


**Fig 1 - Importance of Big Data Processing Using Apache Spark**

ISSN: 3071-3498
PP: 01-16.

*1.2.1. Scalability and Distributed Processing*

Apache Spark is created to run on distributed clusters so it can be scaled horizontally by adding worker nodes in an attempt to increase the amount of data it can process. It is necessary because big data environments may expand to terabytes, or petabytes in size. Spark can easily divide up data and distribute the computation across multiple nodes enabling organizations to compute large volumes of data in parallel. This feature guarantees that performance is not diminished during the upgrowth of data size and complexity of workloads, and thus Spark is a valuable platform that supports enterprise analytics.

*1.2.2. In-Memory Computing for High Performance*

The in-memory computing model is one of the most important benefits of Spark. Spark allows avoiding a large part of the disk I/O operation that is a significant bottleneck in existing disk based systems, such as Hadoop MapReduce, by caching frequently accessed datasets into main memory. This would result in significant savings in run time, especially to iterative algorithms and interactive analytics. Both machine learning and graph processing, as well as real time data analysis use cases, can be heavily utilized using Spark because in-memory processing allows quicker access to data and reduced latency.

*1.2.3. Support for Diverse Workloads*

Apache Spark offers a platform that has a flexible workload support and serves as a single analytics engine to a large variety of workloads. These are batch processing, SQL-based analytics (Spark SQL), machine learning (MLlib), graph processing (GraphX) and stream processing (Structured Streaming). This adaptability decreases the requirement to have several specialized systems installed and maintained, smooths out data architecture and enhances efficiency of operations. Spark allows organizations to have the ability to use the same platform to support several different analytical needs, which enhances productivity and lowers the complexity of the system.

*1.2.4. Fault Tolerance and Reliability*

Spark uses standard in-house fault tolerance to focus on dependable performing in an environment spread out. On lineage-based recovery, Spark automatically views all the lost partitions in case of a node failure without needing to replicate the complete data of the intermediates result. This method contains a high level of robustness and low storage overhead. Fault tolerance will be of the essence in a large cluster where system and network outages are common place, and the Spark implementation assists in maintaining the reliability of jobs and the preservation of data.

*1.2.5. Integration with Big Data Ecosystem*

Apache Spark is also compatible with distributed file system Hadoop Distributed File System (HDFS), data science framework Apache Hive, and Hbase and Apache Kafka on the cloud among other systems. This interoperability enables organizations to utilize available data infrastructure in the adoption of Spark as a state-of-the-art analytics. Accessing various data sources and systems

ISSN: 3071-3498
PP: 01-16.

caused Spark to be a versatile and viable solution to the modern data pipelines and enterprise analysis settings.

### 1.3. Apache Spark: A Performance Study

Apache Spark is one of the brightest distributed data processing frameworks of big data analytics, as it has the ability to deliver high performance, scalable, and unified data processing performance. In contrast to the older frameworks (disk-based, e.g. Hadoop MapReduce), Spark operates within the context of an in-memory computing model that allows intermediate datasets to be re-used across several phases of computational work. The reasoning behind this architectural design is that disk I/O overhead, a significant bottleneck in large-scale data processing systems, is reduced significantly by this design, and enables Spark to achieve significantly lower latency and more throughput across a whole spectrum of applications. This performance study concentrates on the systematic assessment of the execution behavior of Apache Spark to various types of workloads and system set-ups. The study also strives to bring a holistic viewpoint of the performance of Spark to realistic conditions by examining batch processing, iterative machine learning and SQL based analytical workloads. These types of workloads are typical of typical enterprise and research application scenarios, which allows one to draw valuable conclusions about the strengths and weaknesses of Spark in real-world applications. The research highlights the aspect of raw execution time but also resource use, scalability, and system efficiency by giving the multidimensional view of the performance. The major goal of this performance analysis is to understand the way in which the architectural capabilities of Spark e.g. Resilient Distributed Dataset (RDDs), DataFrames, and directed acyclic graph (DAG) execution engine help to enhance performance. All these elements allow Spark to streamline the execution plans and pipeline functioning and reduce the data materialization that should not be achieved. Furthermore, the paper also examines how critical configuration parameters such as executor memory, partitioning strategies and parallelism levels, affect the overall system performance. These factors should be known to achieve good performance in practical deployment. In addition, this performance study is accompanied by a performance comparison with Hadoop MapReduce in order to outline the performance advantages and disadvantages related to the in-memory execution model of Spark. The study offers a useful insight by measuring performance improvements and finding out trends in scalability across the systems, and offer a practical advice on system designing and practitioners. All in all, this section is what defines the rationale behind an elaborate experimental analysis and preconditions the realization of how the Apache Spark may be successfully used to address the performance requirements of emerging big data applications.

## *2. LITERATURE SURVEY*

### 2.1. Hadoop MapReduce Performance Studies

The performance properties of the Hadoop MapReduce system have been immensely studied by various researchers since its launch by Dean and Ghemawat. Their seminal paper introduced the MapReduce programming model as an example of a large-scale data processing, fault-tolerant, and

ISSN: 3071-3498
PP: 01-16.

high-scale model written on commodity hardware. Later studies have shown that MapReduce is very good in large and shamefully parallel batch-processing workloads especially in applications involving log processing, indexing, and offline analytics. But empirical performance measurements have always attributed high job latency and this is mainly because of the data entry requirement of the intermediate data to disk between the map and reduce phases. Such massive use of I/O disk overheads it, imposing throughput and response time performance limits, particularly to workloads where nodes need to be synchronized or have their data shuffled periodically.

### 2.2. Apache Spark Architecture and Performance

Zaharia et al. proposed Apache Spark, an in-memory cluster information processing system, to supply the disk-based system (Hadoop MapReduce) with performance constraints. The main abstraction of Spark, the Resilient Distributed Dataset (RDD), enables the data to be kept in the memory between the iterations, which has reduced disk I/O to the minimum and has boosted speed in the execution as well. Their experimental results showed drastic improvement in the performance, the speed up of up to 100 times on iterative machine learning workloads and up to 10 folds on interactive data analysis tasks. Later research has also delved into the directed acyclic graph (DAG) execution engine of Spark that permits a better allocation of tasks and pipelining of operations. As much as Spark offers major performance advantages, studies have also shown issues associated with memory management such as sensitivity to executor memory configuration, garbage collection overhead, and degradation of performance under memory pressure.

### 2.3. In-Memory Computing Frameworks

Besides Apache Spark, there are other in-memory computing systems that have been put forward and tested in literature. An example of this is Apache Flink, which implements a stream processing model with much emphasis on native streaming and stateful streaming, providing low latency execution with powerful event-time semantics. Apache Ignite, conversely, is oriented on in-memory capabilities of data grid facilities that offer distributed caching, SQL querying services and computing facilities that are closely coupled with set of datasets that are held in memory. Comparative performance analysis has also demonstrated that despite Spark continuing to be a leading general purpose analytics engine, dedicated frameworks like Flink can be faster than Spark with continuous stream processing work loads, and Ignite can be beneficial to applications with high-throughput, distributed in-memory data access requirements. These results indicate that the choice of frameworks depends on the nature of workload and applications.

### 2.4. Gaps in Existing Research

Although there is a large literature on distributed data processing framework, there are still a few relevant gaps in the literature. Majority of the previous researches with a limited range of benchmark workloads or executed in a small scale cluster restrict the generalizability of the research. Moreover, numerous analyses fail to explicitly and mechanically investigate how tuning parameters including memory, levels of parallelism and data partitioning plan affect system performance. Very

ISSN: 3071-3498
PP: 01-16.

little is also done to analyze workload interaction with system architecture and resource constraints in normal deployment environment with different types of workloads e.g. batch, iterative, and streaming workload. This paper will fill these gaps by doing a more thorough and methodical analysis of various workloads, cluster set-ups, and tuning conditions to offer more valuable information about the performance and scalability behaviour.

## *3. METHODOLOGY*

### 3.1. Experimental Setup

The distributed computing cluster (full of worker nodes) where the experimental assessment will take place features a high-speed Ethernet network that will guarantee the communication between the nodes is fast as well as that the information transfer between the nodes will be efficient. All the nodes in the cluster have multi-core processors to enable parallel computation of tasks, main memory (RAM) high enough to support in-memory data processing loads, and local storage to support disk-based operation and maintaining the intermediate data. This hardware layout is developed to resemble a closely similar environment with commodity clusters similar to what one may find in real world deployment of big data thus making the experimental findings more practical in nature. The cluster software stack will be set up to run both Apache Hadoop MapReduce and Apache Spark to allow a direct and equal comparative analysis of the two systems by operating them under the same hardware and network circumstances. Hadoop is set up with the Hadoop distributed file system (HDFS) as its storage level which offers inexpensive fault tolerated distributed file storage and replication. Apache Spark is used on the same top of the HDFS infrastructure, so both frameworks can use the same datasets and avoid the variability caused by the dissimilarity in storage backends. This single storage structure provides data access patterns consistency and meaningful performance comparisons. Moreover, the parameters of system configuration and resource allocation are thoroughly managed to provide the conditions of a stable and reproducible experiments. Execution memory, number of cores on an executor, amount of parallelism, and block size on HDFS are parameters that are optimized after considering the best practices and initial pilot trials. To reduce external interference, and variations in performance, network bandwidth, disk I/O performance and system load will be monitored. Each and every experiment is repeated and average execution time reported to eliminate the effect of fluctuous system variations. Such a controlled experimental setting allows the systematic study of the execution time, resources usage, and scaling nature of both Apache Spark and Hadoop MapReduce on the workloads of different types. The experimental setup through the stable hardware, software and configuration specifications presents a strong base of the comparative strengths and weaknesses of disk-based and in-memory distributed data processing frameworks to realistic cluster conditions.

ISSN: 3071-3498
PP: 01-16.

### 3.2. Workload Description

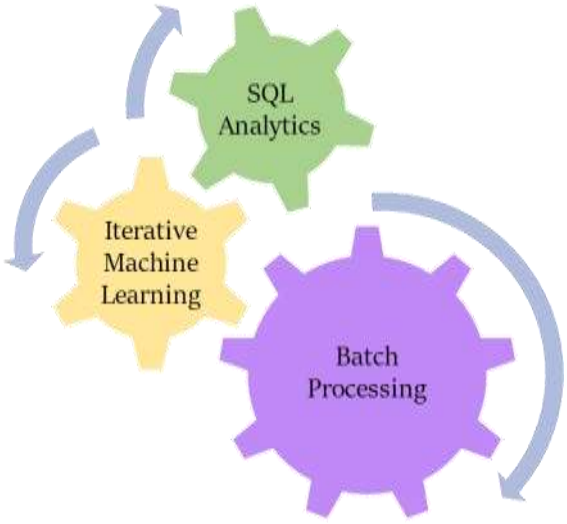


**Fig 2 - Workload Description**

*3.2.1. Batch Processing*

The workloads of batch processing are structured to capture traditional large-scale data processing applications that are a typical case in an enterprise and cloud setting. These loads are mostly the big-volume log processing and extract-Transform-load (ETL), the operations of reading raw data in the distributed storage, using a series of transformation and filtering operations and write the processed results back to storage. This type of workload commonly has an I/O-intensive nature, and focuses on throughput rather than the latency. They are used as an exemplary workload on the performance of Hadoop MapReduce and Spark of processing large, disk-based batch workload to test ingestion volumes of data and shuffle performance at large data volumes, and to measure the overall performance of the job under heavy data volumes.

*3.2.2. Iterative Machine Learning*

The workloads are iterative machine learning workloads that test the load frameworks in a situation where the algorithms repeatedly access and update intermediate datasets with repeated iterations. In particular, k-means clustering and logistic regression are the representative algorithms because of their popularity as data analytics and machine learning engines. These algorithms are prone to repetitive operations on the same data and thus very sensitive to disk I/O overheads and the use of memory. This type of workload is especially effective in rendering positive impacts of using in-memory processing in Apache Spark because caching intermediate products in memory can be a great way of saving a lot of time when compared to using a disk-based platform to save in-memory caching intermediate products. The efficiency of the frameworks in supporting iterative patterns of computation and handling data available as memory residence is the measure of performance in this category.

ISSN: 3071-3498
PP: 01-16.

*3.2.3. SQL Analytics*

`SQL analytic workloads are modelled on interactive and ad hoc analytical queries that are commonly executed in applications of data warehousing and business intelligence. These workloads are characterized by elaborate aggregation, filtering, as well as, the multi-table join operations computed with the Spark SQL. The questions will mimic real-world analytical phenomena, including summaries of large datasets and aggregating two or more data to create insights. This category focuses on the query execution latency and optimization efficacy such as predicate pushdown and query plan optimization. SQL analytics workload testing drivers can be used to understand the appropriateness of Apache Spark in interactive analytics, as well as attribute its performance benefits to a traditional batch-oriented processing framework in the case of structured query loads.

### 3.3. Performance Metrics

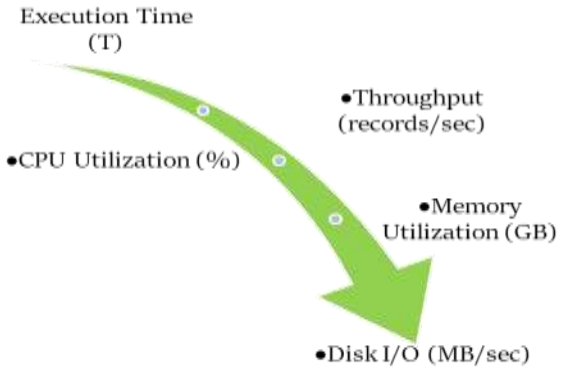


**Fig 3 - Performance Metrics**

*3.3.1. Execution Time (T)*

Execution time is the number of moments that it takes to run a particular workload, which is the time it takes to submit a job until a job is successfully completed. This is one of the main metrics that dominate the system performance and are utilized to compare the efficiency of Apache Spark and Hadoop MapReduce in terms of the workload type. Reduction in execution time is an indication of improved use of system resources and more effective scheduling, access to data and calculation. Execution time socialization allows to estimate directly the end-to-end performance gains obtained by in-memory processing and optimal execution engines.

*3.3.2. Throughput (records/sec)*

Throughput is a measure of how fast the system processes records in a given unit of time in example of an instance per minute. Using this measure is especially applicable to batch processing workloads, at which point the capability to process large amounts of data effectively is paramount. Increased throughput implies increased capacity of data processing and implies the flexibility of the system to scale as the amount of data is increased. The throughput has been employed to determine the effectiveness of the different frameworks in maintaining high processing rates in large workloads.

ISSN: 3071-3498
PP: 01-16.

*3.3.3. CPU Utilization (%)*

CPU utilization denotes percent utilization of the available processor capacity in the existence of workload. This value tells us the level of efficiency on which each of the frameworks is using its computational resources. High CPU utilization can be an indication that there is good parallelism and processing-intensive workloads but low CPU utilization can be a potential indicator of bottlenecks at the I/O or poor scheduling. By examining CPU usage, one will know whether defining performance constraints are based on a lack of computation, delays during data access, or ineffective allocation of tasks.

*3.3.4. Memory Utilization (GB)*

Memory utilization Ratio of the size of the main memory that the workload has used throughout its execution period, such as memory used to store datasets, processing intermediates, and framework overhead. This measure is especially valuable to in-memory storage, like the Apache Spark, where the capabilities of the memory management system are directly related to performance. Using memory utilization can be used to measure the trade-off between memory consumption and performance improvement, as well as facilitating in determining memory pressure, disk spilling, and overhead of garbage collection.

*3.3.5. Disk I/O (MB/sec)*

Disk I/O is given in megabytes per second to measure the rate at which data is read or written into disk during the execution of the workload. This measure is essential to realize the effect of disk usage by processing in Hadoop MapReduce and how much disk access can be reduced by in-memory computation in Apache Spark. One metric that can be used to identify overreliance on intermediate data materialization and under relyance on memory caching is large disk I/O rates and small disk I/O rates, respectively. The disk I/O is also useful to analyze data movement patterns and its role in the overall performance.

**3.4. Experimental Procedure**

The aim of the experimental procedure is to provide repeatability and statistical reliability and general performance checks and balances in the diverse system configurations. The individual workloads are run on at least two occasions after being subjected to same experimental conditions in order to counter variability caused by the background system activities, network ups and downs, and the operating system scheduling effects. At the end of every experiment run, execution logs and performance statistics at the system level are gathered in detail. The average time spent in the execution is calculated over the repeated runs and used as the major performance metric, which minimizes the effect of extremes and temporary deviations. Along with average values, variability is observed to make sure that the values are consistent, as well as to confirm that the environment of the experiment remains stable. In order to investigate the effect that system configuration has on performance, a sequence of controlled experiments is done by systematically sweeping important tuning parameters. The executor memory is tuned on several levels to assess the sensitivity of the

ISSN: 3071-3498
PP: 01-16.

Apache Spark and Hadoop MapReduce systems to memory allocation and observe the impact of memory availability to the performance of caching, garbage collection, and disk spill. The data partition number is also diversified to examine its effect on the task parallelism, load balancing, and its performance shuffle. An appropriate partitioning is essential to obtain optimal use of resources, and such experiments are used to discover the most effective partition scheme with various types of workloads. In addition, the size of the clusters will be scaled by changing the amount of worker nodes in operation in order to assess the scaling properties of the two structures. This facilitates the examination of the variation of performance measures like the execution time, throughput and resource usage with the provision of extra computation and storage units. Scalability experiments give an understanding of how well the frameworks can utilise increased parallelism as well as how communication and coordination overhead in larger size clusters can be managed. Any change of configuration is also done under controlled conditions and a single parameter is changed at a time and all the other parameters kept fixed so as to separate the effect of each individual factor. Such systematic experimental approach allows one to understand the performance tendencies and trade-offs in more detail and offers a solid background to compare disk-based and in-memory distributed data processing structures.

### 3.5. Mathematical Performance Model

In a bid to quantitatively compare and contrast the performance of Apache Spark and Hadoop MapReduce, this paper uses common metrics of performance modeling tools such as speedup and scalability effectiveness. These mathematical indicators give a systematic and objective ground in the evaluation of the relative performance improvements realized by in-memory processing besides the evaluation of the efficiency of the use of the extra computation resources by the system. The relative performance improvement of Apache Spark over Hadoop MapReduce with the identical load and data set is achieved by speedup. Speedup in this research paper describes how many times faster Hadoop MapReduce in comparison with Apache Spark runs. That is, speedup can be computed by dividing the total time taken by MapReduce by the execution time taken by Spark. The speedup value exceeding one points to the fact that Spark has better performance than MapReduce, and the bigger the number of the values the better Spark performs. The metric is a direct measure of in-memory execution benefit and efficient scheduling of tasks in Spark, especially where the workloads are iterative and interactive. Speedup is calculated in every category of workloads and configuration setting in order to make a detailed comparison of the various experiments. Scalability is tested with the efficiency metric which identifies the effectiveness of the system to add more cluster nodes. The ratio of the execution time of single node to the product of number of nodes and execution time of N nodes is known as the efficiency of the product. In actual practice, efficiency is computed by dividing the time of execution on a single node by the number of nodes multiplied by the time of execution on multi-node. This measure indicates the degree to which the measured performance is linear and ideal. Efficiency of near perfection means that the performance increases proportionately to the added resources which means an efficiency value of almost one. The decrease in the efficiency values show diminishing returns because of overheads in communication,

ISSN: 3071-3498
PP: 01-16.

synchronization, skew of data and contention of resources. The speedup and efficiency measures can be regarded as complementary views about system performance. Whereas speedup shows the relative improvement between various frameworks, efficiency provides an assessment of the behavior of the system in terms of scale as the number of clustering nodes goes up. These mathematical performance models can be used to interpret with rigor and systematicity experimental outcomes and elucidate in deeper detail the trends in performance, trade-offs and scalability constraints in distributed data processing systems.

## *4.RESULT AND DISCUSSION*

### 4.1. Comparative Execution Time

**Table 1: Comparative Execution Time**

| Workload Type | Spark Time as % of MapReduce | Execution Time Reduction (%) |
|---|---|---|
| Batch ETL | 37.50% | 62.50% |
| k-Means | 11.10% | 88.90% |
| SQL Analytics | 31.60% | 68.40% |

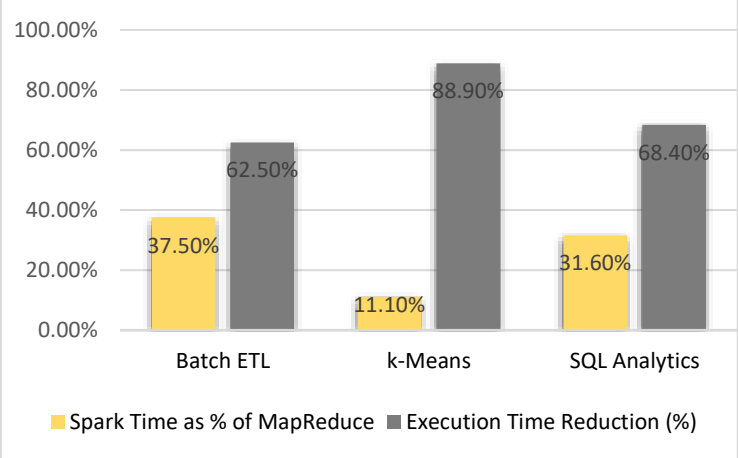


**Fig 4 - Comparative Execution Time**

*4.1.1. Batch ETL*

In the batch ETL workload, the execution time on Apache Spark is equal to 37.5 compared to MapReduce execution time which would represent a reduction of 62.5 in the total job time. This is the huge advancement that proves that Spark minimizes overhead of disk I/O by processing data in-memory and providing more effective task allocation. Despite the opportunities offered by the Hadoop MapReduce framework to handle batch workloads of ETL more effectively, the obtained results do show that Spark can also provide significant performance improvements over its Hadoop counterpart by reducing materialization of intermediate data and optimization of its data flows due to the directed acyclic graph model of execution. The improvement that is observed indicates that Spark is effective even in terms of I/O-intensive batch workloads.

ISSN: 3071-3498
PP: 01-16.

*4.1.2. k-Means*

The largest improvement in performance of the k-Means clustering workload is the most notable as Spark only needs 11.1% of the execution time of MapReduce, which is typical of the execution time reduction of 88.9%. This finding greatly echoes the benefits of in-memory processing to iterative algorithms whereby one has to access the same dataset repeatably throughout multiple iterations. In MapReduce, the intermediate results are stored out to disk at the end of every iteration which creates significant I/O overhead and latency. In comparison, the fact that Spark has a memory caching library to hold data enables consecutive data sets to use previously cached data, thus, saving a lot of time in execution and proving Spark to be a better fit in iterative machine learn.

*4.1.3. SQL Analytics*

In case of SQL analytics workloads Spark takes an equal time of 31.6 percent of MapReduce, and it is 68.4 percent lesser than the execution time. This has been enhanced majorly because Spark SQL has a highly optimized query executable engine with such features as being able to provide cost-based queries optimization, optimal join schemes, and in-memory calculation of intermediate output. These capabilities allow Spark to run complex aggregation and join queries faster than the traditional SQL engines using MapReduce. These findings show that Spark is especially efficient in interactive/analytical query processing, in which the latency and the query response time are of utmost importance.

**4.2. Resource Utilization**

According to the experimental findings, it can be stated that there are evident differences in the patterns of resource utilization between Apache Spark and Hadoop MapReduce due to their radically different models of execution. Spark invariably shows an increased amount of memory usage in all workload types that is attributed to the wide use of in-memory caching of data and the persistence of intermediate data sets. Via caching on access data in main memory, Spark vastly aids the necessity to read data stored on disk repeatedly, and over an alphabet of calculation steps. This extra memory usage is a design decision aimed at having Spark realise significant performance gains, specifically in relation to iterative and interactive workloads. Conversely, Hadoop MapReduce has an extensive use of disk-based storage to store intermediate results hence a relatively low use of memory but highly intensive disk input/output operations. MapReduce uses local disk between the map and reduce stages to store intermediate data and between job stages to store intermediate data, which leads to the eventual use of the disk read and disk write frequently. This action puts up huge I/O overhead and job latency, particularly to workloads, which consist of many processing stages or frequent data access. The experimental data prove the idea that MapReduce has significantly more disk I/O rates compared to Spark, thus showing a significant reliance on disk operations to materialize intermediate data. One of the factors that have contributed to the better performance of Spark is the decrease in disk I/O. Through disk access reduction and utilization of the data stored in the memory, Spark reduces I/O wait time and selective processing of tasks. This will enable the CPU resources to be put to better use since processors will use less time of waiting as the disk operations

ISSN: 3071-3498
PP: 01-16.

are made. As a result, Spark can reach greater efficiency in the entire system, although it will take more memory resources. These results illustrate that there is a significant trade-off between the memory and disk I/O of distributed data processing systems. Although Spark will be memory intensive to enable in-memory caching, the rise in memory usage results in significant decrease in disk reads and the total time of execution. Based on the system design, the findings indicate that the memory is an important resource that must be provisioned so as to make the most out of the performance advantages of Spark. In general, the resource utilization analysis shows that in-memory implementation of execution model in Spark offers a more-efficient usage of system resources to contemporary data analytics tasks than the traditional disk-based MapReduce processing.

**4.3. Scalability Analysis**

The experiments of scalability will test the ability of Apache Spark to utilize more cluster resources with the increase of the number of worker nodes. The findings indicate that Spark can essentially linearly scale batch processing and SQL analytics loads, which implies that the time to complete the task duly reduces in almost linear relation with the increase in computational resources. This is indicative of the fact that these workloads are well-parallelized and are high-performance in terms of higher task-level parallelism and, hence, Spark is well positioned to scale out data partitions and computation across many nodes. The observed near-linear scaling is due to the fact that Spark can effectively coordinate task scheduling and data distribution, and communication between nodes when workloads are less constrained by memory and synchronization. Contrastingly, machine learning tasks that are memory-intensive and include k-means clustering and logistic regression have diminishing returns with respect to cluster size. Performance is still increasing with the number of nodes, but this decreases with a cluster size. This is mainly because of the high communication and memory overheads of distribution and synchronization of large in-memory sets with the growth of nodes. The higher the number of nodes, the higher the cost of data shuffling, broadcast, and barriers to synchronization, made less efficient is further parallelism. Also, memory pressure is augmented when intermediate data and model parameters are replicated or shared across nodes. This may cause a greater level of garbage collection, diffusion, and possibly data spillage to disk contributing to a poor performance level. Such impacts restrict the scalability advantages of iterative machine learning workloads which are based on intensive communication and recurrent accessibility to shared state. As the scalability analysis notes, although Spark can successfully utilize more resources to scale to embarrassingly parallel and query-based workloads, due to system-level overheads, its scalability with more memory and communication intensive workloads is limited. These findings show that cluster sizing and workload-aware resource provisioning are crucial. Increase in the number of nodes is not necessarily proportional to performance, especially with iterative machine learning jobs. In general, the results indicate that Spark has good scalability features, though to achieve an optimal performance, the aggressiveness of the workload and cluster size, memory capacity, and communication overhead need to be balanced.

ISSN: 3071-3498
PP: 01-16.

### 4.4. Discussion

The experimental findings are very clear to demonstrate the fact that Apache Spark is very appropriate with the respect to productive execution model of iterative, interactive, and the task scheduling mechanisms. The significant cuts in the execution times of the machine learning and SQL analytics workloads indicate the effectiveness of Spark in managing other computing operations effectively by reusing intermediate datasets. Spark dramatically reduces reinstatement of disk I/O operations which is a significant contributor to latency in Hadoop MapReduce systems using traditional systems. Such an ability makes Spark particularly ideal to workloads that need frequent data reutilization, quick rehabilitation, and low-latency query processing. But, the findings also point to the fact that to have an optimum performance with Spark, the system needs to be tuned and configured closely. Executor memory distribution is a crucial factor in defining the effectiveness of Spark to store datasets and handle intermediate output. The variable memory may be insufficient and contribute to the frequent usage of a garbage collection, the possibility of memory fragmentation, and data spilling on disk, which adversely affect the performance and eliminate the advantages of in-memory processing. In some severe situations, poor memory provisioning can cause out of memory errors, which causes job failure and low system reliability. Another important issue in Spark performance is partitioning strategy. Poor partitioning may cause skew of data, i.e. one task may operate on a disproportionately large partition hence loading balance and increasing the tasks completion time. A too large partition will put a strain on the memory of specific executors and a too small partition may lead to excessive scheduling overhead and inefficient performance. Thus choosing the optimal number of partitions and achieving even distributions of the data is crucial to simply gaining as much parallelism as possible and reducing resource contention.They also highlighted discussion shows the trade-offs which in in-memory computing frameworks involve. Although, Spark provides important performance benefits, the improvement is achieved at the expense of accelerated memory demand and sensitivity to configuration variables. This implies that the organizations implementing Spark should take time to calculate cluster memory capacity and invest in performance tuning to achieve maximum benefits. In general, the results highlight that Spark can still be better used with the current analytics workloads, but this is heavily dependent on the correct provisioning of resources, memory management, and workload-sensitive configuration.

## *5. CONCLUSION*

The paper has given an in-depth and systemic performance analysis of Apache Spark as a distributed big data processing system, and specifically contrasted the framework with Hadoop MapReduce on a wide range of types of workloads. The experimental test was to simulate real cluster settings and it involved batch processing, iterative machine learning as well as SQL-based workloads in analysis. The findings illustrate clearly that Apache Spark out-perform Hadoop MapReduce regularly in all the types of workloads that were tested, and that more noticeable performance improvements are found in the case of iterative and interactive workloads. The findings are a strong indication that in-memory computing paradigm of Spark can be used in modern applications that consume large amounts of data. Empirical evidence proving the paramount importance of in-

ISSN: 3071-3498
PP: 01-16.

memory data processing is a major contribution of the current study that allowed defining the minimum execution time and disk I/O overhead reduction. Spark is able to store intermediate datasets in memory and by doing so, it greatly alleviates repeated disk reads which is a significant performance bottleneck of traditional disk based processing models. Moreover, the directed acyclic graph (DAG)-based execution engine provided by Spark allows even more effective scheduling and pipelining of computing phases, which also leads to the smaller resource usage and the shorter lead times. A combination of these architectural characteristics enables Spark to achieve a large performance benefit, especially when used on a workload in which repeated data access, iterative computation, and intricate analytical queries comprise a significant portion. The scalability analysis also indicates that Spark can attain relative linear scale in performance improvement in batch processing and SQL analytics workloads with increase in cluster size. It means that Spark could efficiently utilize extra computation resources in order to enhance throughput and decrease the execution time. Simultaneously, the paper finds significant constraints to machine learning workloads that are memory-intensive and where communication overheads, memory pressure, and the costs of synchronization results in decreasing returns as the cluster scale is increased. These findings underscore cluster sizing and performance tuning due to workload variability are essential to the provision of optimal scalability. Although the general findings are highly in favor of Spark, the issue of memory issues and complexity of configuration has been ventured too. The optimum performance is only possible through proper tuning of executor memory, partitioning, and parallelism. Poor configuration may cause the collection overhead of garbage to go up, spill-over of memory, and even memory failure of the job because of out-of-memory problems. These results indicate that Spark offers great benefits in terms of performance but it also requires more proficiency in system optimization, as well as the resources management. This analysis will be extended to heterogeneous and cloud-native systems in the future, where resource elasticity, overhead of virtualization, and variability in the network are other performance factors. More so, the merging of adaptive and intelligent resource control techniques, including dynamic executor scaling and automatic configuration fine-tuning is also a promising future of usability and robustness of performance. All in all, this paper adds significant value to the performance aspect, scaling attribute, and deployment factors of Apache Spark, which can further continue to gain popularity as an open-source data analytics solution used to handle vast amounts of data.

## REFERENCES

[1] Dean, J., & Ghemawat, S. (2004). *MapReduce: Simplified data processing on large clusters*. Proceedings of the 6th USENIX Symposium on Operating Systems Design and Implementation (OSDI).

[2] Dean, J., & Ghemawat, S. (2008). *MapReduce: Simplified data processing on large clusters*. Communications of the ACM, 51(1), 107–113.

[3] Zaharia, M., Chowdhury, M., Franklin, M. J., Shenker, S., & Stoica, I. (2010). *Spark: Cluster computing with working sets*. Proceedings of the 2nd USENIX Workshop on Hot Topics in Cloud Computing (HotCloud).

[4] Zaharia, M., Xin, R. S., Wendell, P., et al. (2016). *Apache Spark: A unified engine for big data processing*. Communications of the ACM, 59(11), 56–65.

[5] Shi, J., Qiu, Y., Minhas, U. F., et al. (2015). *Clash of the titans: MapReduce vs. Spark for large-scale data analytics*. Proceedings of the VLDB Endowment, 8(13), 2110–2121.

ISSN: 3071-3498
PP: 01-16.

[6] Lin, J., & Dyer, C. (2010). *Data-intensive text processing with MapReduce*. Synthesis Lectures on Human Language Technologies, Morgan & Claypool.

[7] Zaharia, M., Das, T., Li, H., et al. (2013). *Discretized streams: Fault-tolerant streaming computation at scale*. Proceedings of the 24th ACM Symposium on Operating Systems Principles (SOSP).

[8] Carbone, P., Katsifodimos, A., Ewen, S., et al. (2015). *Apache Flink: Stream and batch processing in a single engine*. IEEE Data Engineering Bulletin, 38(4), 28–38.

[9] Katsifodimos, A., Tzoumas, K., & Markl, V. (2016). *Flink: Batch and stream processing in a single engine*. IEEE ICDE.

[10] Abramova, V., & Bernardino, J. (2013). *NoSQL databases: MongoDB vs Cassandra*. Proceedings of the International Conference on Computer Science and Software Engineering.

[11] Abadi, D. J., et al. (2016). *The design and implementation of modern column-oriented database systems*. Foundations and Trends in Databases.

[12] Akil, B., Zhou, Y., & Röhm, U. (2018). *On the usability of Hadoop MapReduce, Apache Spark & Apache Flink for data science*. arXiv Technical Report.

[13] Dolev, S., Florissi, P., Gudes, E., Sharma, S., & Singer, I. (2017). *A survey on geographically distributed big-data processing using MapReduce*. IEEE Communications Surveys & Tutorials.

[14] Singh, P., Singh, S., Mishra, P. K., & Garg, R. (2019). *RDD-Eclat: Approaches to parallelize Eclat algorithm on Spark RDD framework*. Future Generation Computer Systems.