Performance Comparison of Hadoop-Spark Cluster for Performing Sentiment Analysis on Big Data

Arun Karthick C[1], Skanath Kumar P S[2], Sallagarige Rahul[3], Dr. A. Nagaraja Rao[4]
[1,2,3]M. Tech. CSE (Big Data Analytics), VIT, Vellore, India
[4] Associate Professor, VIT, Vellore, India
doi.org/10.64643/IJIRTV12I11-186029-459

Abstract— **The explosive growth of user-generated content on the web has driven the need for efficient sentiment analysis at a massive scale, demanding robust distributed computation. Recent advancements in big data technologies especially Apache Hadoop and Spark enable parallel processing and rapid anal- ysis of voluminous text datasets. In this study, we configure, evaluate and benchmark single-node and multi-node cluster deployments leveraging batch-oriented big data processing to classify sentiment in large movie review corpora and compare the execution time from single node versus Multi Node cluster. Our approach centers on the configuration, orchestration, and comparative analysis of different cluster setups using Hadoop and Spark as distributed batch frameworks. Experiments examine effects on performance, data locality, and scalability, providing practical insights for real-world system architects. Our results show that while cluster overhead may negate parallel speedup for small workloads, multi-node deployments consistently outperform single-node setups when processing large-scale data. From our observations we were able to interpret that the Multi Node cluster performed 30% more efficiently than the Single Node. The study also investigates system fault tolerance during node failures and findings provide practical insights on cluster configuration and workload suitability for scalable sentiment analysis.**

Index Terms—**Hadoop, HDFS, Map-Reduce, Spark, Single- Node, Multi-Node Cluster, Distributed Storage, Parallel Computing, Namenode, Datanode, Sentiment Analysis, Fault Tolerance, Big Data Analytics, Cluster Configuration, Sentiment Analysis.**

I. INTRODUCTION

In today's digital world, data is growing at an unprecedented rate, and processing large amounts of it efficiently has become a major challenge. Traditional systems often struggle to keep up, leading to slow performance, delays, and system overloads. This is where distributed computing comes in, offering a way to break down large tasks into smaller chunks and process them simultaneously across multiple machines. Recent advances in big data analytics and distributed computing frameworks have revolutionized how organizations handle massive datasets. As Haddad et al. (2024) demonstrate in their intelligent sentiment prediction approach, the integration of batch big data analytics using frameworks like Hadoop and Spark enables powerful text processing capabilities at scale [1]. Their work, achieving 96% accuracy in sentiment classification, highlights the significant potential of combin- ing distributed storage with machine learning for real-world applications. For this project, we set up a Hadoop–Spark multi- node cluster designed to handle big data efficiently. Our cluster consists of one name-node and two data-nodes, working together to store and process data in a distributed manner using Hadoop's HDFS (Hadoop Distributed File System) and MapReduce framework. Most of the college academics constrain themselves to the single node architecture where the name node and the data-nodes are in the same single device. This poses the problem of not understanding the current state of the application. Also, single-node devices are capable of processing only a small size of the dataset, and they fail to process a large data set compared to the resources provided by the single-node. This has been clearly proved in our study. To pre-process such a large dataset, we require the power of distributed processing which is provided by the hadoop multi node cluster configuration. The study comprises various comparisons between a single node and a multi node cluster, for different file sizes and their execution time are recorded. To properly evaluate the

performance of the cluster, we have used the application of Sentiment Analysis of a movie review to classify the reviews as positive or negative. For this application of Sentiment Analysis, Spark's Machine Learning Libraries are used using the scala API.

There were many challenges faced in every step of the process, since the setting up of a cluster with Hadoop, HDFS, YARN, Spark and Master-Worker organization, is not an easy task. The documentation provided by Hadoop [2] and Spark [3] were the major support in configuring the desired cluster. We have chosen Spark along with Hadoop because it has the following advantages. Recent studies have shown that such configurations can significantly improve processing times, with Apache Spark running applications up to 100x faster in memory and 10x faster on disk compared to traditional Hadoop MapReduce [4]. Additionally, Apache Spark is in-corporate into this configuration to take advantage of its MLlib machine learning library and in-memory processing ca- pabilities, which greatly speed up complicated data processing workloads and iterative computations beyond the conventional MapReduce architecture. The ability of Spark to do batch, interactive, and machine learning tasks within a single cluster is what makes it so well-known. Spark's scalability is one of its primary characteristics, allowing the cluster to have a" n" number of nodes. Another main feature in Spark is language flexibility so Spark developers can use (API) in Scala, Python, Java and R programming languages [5]. Scala is used in this study for submitting the job to the spark executor with all the instructions.

The contemporary landscape of distributed computing has seen remarkable developments in performance optimization and cluster configuration. Research by Zhang et al. (2024) on multi-node Hadoop clusters in cloud environments demon- strates that proper configuration can lead to substantial per- formance gains, particularly when comparing local versus cloud-based deployments [6]. More importantly, this project highlights how distributed computing with Hadoop and Spark can make complex data processing tasks more manageable, opening the door for real-world applications like business man- alytics, social media monitoring, and large-scale data mining. The paper is organized as follows: Abstract, Keywords, In-troduction, Literature Review, Hadoop Spark Cluster Working Environment, Sentiment Analysis of movie review using the Cluster, Results and performance analysis, Conclusion, Future Scope.

II. LITERATURE REVIEW

In this section, we will present previous literature studies that have been conducted in the field of Big Data Analytics, Distributed Processing using Hadoop and Spark. The field of big data processing has evolved through foundational models like MapReduce [7], which simplified parallel computing for distributed systems and led to the development of Hadoop. More recent research focuses on integrating machine learning and real-time analytics with big data frameworks. Notably, Haddad et al. (2024) [1] proposed a sentiment prediction model that combines batch and streaming analytics using deep learning techniques to analyze social media sentiment in real time. This reflects the increasing need for hybrid mod-els capable of processing both historical and real-time data. Additionally, Sokolova (2018) [8] explored challenges in text classification, which is crucial for handling large volumes of unstructured text data, while Tang et al. (2022) [9] highlighted the versatility of Apache Spark in supporting both big data processing and machine learning. The integration of cloud computing with big data processing has been an important development, as discussed by Sandhu (2022) [10], which presents solutions to challenges such as storage, security, and performance in cloud-based systems. Meanwhile, Zarei et al. (2022) [11] reviewed the evolution of Hadoop, examining its current role and challenges in modern big data environ- ments, particularly its integration with cloud platforms and streaming analytics. These studies collectively demonstrate the growing complexity of big data systems, mov- ing from foundational models like MapReduce and Hadoop to more ad- vanced hybrid frameworks that incorporate real-time analytics, machine learning, and cloud computing to address scalability and performance challenges. This combination helps speed up processing, ensures that no single machine becomes a bottleneck, and provides fault tol- erance in case of failures. Instead of relying on one computer to do all the heavy lifting, we spread the workload across multiple machines, making the entire system more efficient and reliable [12].

To demonstrate the power of our cluster, we tested it with a sentiment analysis task, where we processed

large amounts of text data to analyse emotions and opinions. The integration of big data, machine learning (ML), and senti- ment analysis has revolutionized modern business strategies, enabling companies to extract deeper insights from customer interactions [13]. The use of sentiment analysis for larger datasets would be very important for the businesses to know their customer satisfac- tion and accordingly decisions could be made to improve the customer satisfaction. Lim and Park [14] address the performance challenges in heterogeneous single-board computer (SBC) Hadoop clusters, which are often formed by combining different generations of Raspberry Pi devices. They redesign the Hadoop YARN framework with two scheduling approaches master-driven and slave- driven and improve Application Master task alloca- tion by using precise and detailed node resource information. Ex- perimental results demonstrate significant performance im- provident compared to native Hadoop, achieving up to 2.55× faster execution for I/O-intensive workloads and 1.55× faster performance for CPU-intensive tasks. These enhancements make the system more efficient and suitable for resource- constrained environments.

Bhathal and Singh [15] investigate vulnerabilities in the Hadoop framework, which is widely used for storing and processing Big Data in domains such as healthcare, social media, and other industries. They highlight that Hadoop's flexible and distributed architecture can intro- duce security risks, and they demonstrate actual attacks in an experimental environment to evaluate the impact. The results show that such attacks can degrade system performance, leading the authors to recommend a defense-in-depth strategy to strengthen data protection and mitigate risks, particularly in heterogeneous setups. Kalia and Dixit [16] focus on problems when Hadoop MapReduce runs in a heterogeneous cluster, where the default data locality feature does not work well and causes slow work. They gave a new idea with KNN-based scheduler that mixes speculative prefetching and grouping of map task output before reducer start. This helps less network congestion and make processing faster. They test this with workloads like Wordcount, Random Text, Random Num, and Sort, and results show better time and performance compared to the default scheduler in a system with different node capability.

Sundara Kumar and Mohan [17] brought a method called Map Reduce Scheduling-Based Non-Dominated Sorting Ge- netic Algorithm (MRSNSGA) for better big data analytics performance. This method mixes genetic algorithm scheduling and data replication in Hadoop, so resource use is better, node availability more stable, and process speed increases. Simulations show 30 to 35 percent faster data processing and 24 to 30 percent better solution choice compared to old methods, and also reduce delay and time to access data in distributed setup. Sundara Kumar et al. [18] improved big data analytics more by changing Apriori algorithm inside Hadoop MapReduce. They added multiplied-fixed-pass combined counting (MFPC) and average time-based dynamic combined counting (ATDFC), which skipped pruning in some passes so it runs quicker. This design makes execution faster without losing result correctness. In their test on a multi-node Hadoop cluster, this method saves 84 to 90 percent execution time com- pared to traditional fixed-pass combined counting (FPC) and dynamic pass combined counting (DPC) methods. Because of this, frequent itemset mining work becomes much quicker, so in big dataset mining jobs, it gives strong benefits. Building upon scheduling research, Xu et al. [19] introduced resource- aware scheduling strategies for heterogeneous Hadoop clusters, allowing jobs to be dynamically assigned based on node capacity, further improving fairness and throughput. Similarly, Chen et al. [20] provided a hybrid workload analysis com- paring Hadoop MapReduce and Spark, showing that Spark outperforms Hadoop in iterative and mixed workloads while Hadoop remains efficient for batch-heavy jobs.

III. HADOOP SPARK CLUSTER WORKING ENVIRONMENT

A. Ease of Use

By leveraging Hadoop's HDFS and MapReduce, data can be automatically split into smaller chunks and distributed across multiple Data Nodes, enabling parallel processing and effi- cient storage. This architecture enhances computation speed, optimizes resource utilization, and ensures high throughput, particularly when working with large-scale datasets. Addi- tionally, the integration of Apache Spark into the cluster significantly improves ease of use for iterative and machine learning workloads.

B. Single Node Vs Multi Node Cluster

A single-node Hadoop cluster is essentially a one-machine setup where all the core components of Hadoop, like HDFS, MapReduce, and YARN, run on the same computer. While this setup is great for testing, learning, or small-scale projects, it has its limits. Since everything is running on just one machine, it can quickly become overwhelmed when processing large amounts of data. There's also no backup in case of failure, which means if the single machine crashes, everything comes to a halt.

A multi-node Hadoop cluster takes things to the next level. Instead of relying on just one machine, it spreads the workload across multiple machines (or nodes), which helps balance the data storage and processing tasks. With more nodes, Hadoop can store and process much larger datasets at once, making it perfect for handling real-world, data-heavy tasks. Adding more nodes to the system increases both storage and processing power, so as the volume of data grows, the cluster can grow with it. What makes the multi-node cluster truly powerful is fault tolerance. In a multi-node cluster setup, if one node fails, the system keeps running without missing a beat. Data is replicated across different nodes, so there's always a backup. This contrasts with a single-node cluster, where a failure means everything comes to a halt.

C. Configuration of Hadoop Spark Cluster

A distributed computing environment was established using Apache Hadoop and Apache Spark. The cluster consisted of one master node and multiple worker nodes, connected via Gigabit Ethernet and a five-port switch. The network was established using SSH Protocol (Secure Shell Protocol). The master node was configured to run critical Hadoop services: the Name Node (responsible for filesystem metadata management) and the YARN Resource Manager (for cluster resource allocation). Each worker node hosted a DataNode (for HDFS data storage) and a YARN Node Manager (for managing computation containers). Spark 3.5.4 was deployed across the cluster. Sentiment analysis jobs were initiated in client mode using the Spark Scala shell, such that the driver program executed on the client machine, while ex- ecutors were dynamically allocated on worker nodes by YARN. All Hadoop and Spark config- uration files (e.g., core-site.xml, hdfs-site.xml, yarn-site.xml, spark-env.sh, spark-defaults') were set ac- cording to best practices for distributed data processing. After successful configuration of the Hadoop and Spark, and after starting all the shell scripts of the Hadoop's and Spark's configuration files on the cluster, the following daemons were running in each of the Master and Data Nodes respectively as shown in the Table I.

Table I Daemons running on Master and Worker Nodes

Master Node	Data Node
Name-Node	Data-Node
Secondary Name-Node	Node Manager
Resource Manager	Workers (Spark)
Spark Submit	
Master (Spark)	

D. Spark - using YARN Cluster Resource Manager - Archi- tecture

The overall architecture and data flow including the roles of the Spark driver, YARN Resource Manager, Node Manager, HDFS Name Node, and Data Nodes are illustrated in Figure 1. This figure highlights the orchestration of distributed compu- tation and storage across the cluster during any job execution.

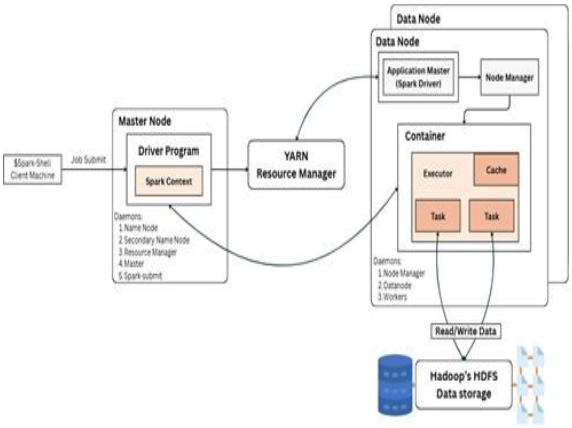

Fig. 1. Architecture of Hadoop-Spark Cluster

The above architecture as shown in Figure 1 was configured and implemented, to perform the Sentiment Analysis for a movie review, with a very large dataset or a Big Data. The client submits the job through the interactive Spark Shell which runs using Scala API. The instructions were given step by step as followed in the next section of Implementation. The Spark shell would be running in the Master Node. Then the Master node starts the spark session and

creates the driver program, which in turn initiates the Spark Context (part of the spark architecture). Parallelly the YARN starts allocating the resources to the available data nodes or the worker nodes. The job is sent to the data nodes which first initiates an application Manager (AM) in one of the data nodes. The Application Manager calculates the required number of resources or con- tainers to execute the job and negotiates the same from the YARN. Then Application Manager gives the instructions to Node Manager about the containers which in turn allocates the resources to perform the task. The Executor, which is a part of the Spark Worker Node, carries out the given tasks and sends the output to the driver program or the Master Node. Then the Master node would aggregate the results to get the final output. Spark would carry on the task using the RDD (Resilient Distributed Datasets) to perform the actions and transformations across the nodes. The input data is read from the Hadoop's HDFS.

Spark is a fast and general processing engine compatible with Hadoop data. It can run in Hadoop clusters through YARN or Spark's standalone mode, and it can process data in HDFS, HBase, Cassandra, Hive, and any Hadoop Input Format. It is designed to perform both batch processing (similar to MapReduce) and new workloads like streaming, interactive queries, and machine learning. Specifically, to run on a cluster, the Spark Context can connect to several types of cluster man- agers (either Spark's own standalone cluster manager, YARN or Kubernetes), which allocate resources across applications [3].

IV. SENTIMENT ANALYSIS OF MOVIE REVIEW USING THE CLUSTER

All data processing logic was implemented in Scala using Spark's RDD and Data Frame APIs. The workflow comprised:

- Data Ingestion: Reading input data from HDFS into distributed memory structures.
- Preprocessing: Tokenization, stopword filtering, and fea- ture vectorization.
- Model Application: Sentiment classification was per- formed using machine learning algorithms such as Logis- tic Regression, leveraging Spark MLlib. The model pro- duced polarity labels (positive/negative) and probability scores for each record.
- Aggregation: Results were transformed and aggregated to generate the output with prediction, accuracy and execution time.

From Figure 2, we can see the process flow for performing the Sentiment Analysis on Movie Review Dataset, using the Configured Cluster.

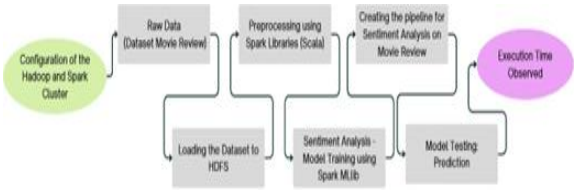

Fig. 2. Process Flow Chart

A. Creating a network and establishing communication among the devices

In order to ensure seamless communication among the devices, it is necessary to set a stable and secure network. For this process, we have used SSH (Secure Shell) Protocol. SSH keys were generated and shared among all the devices that took part in the network. By this, the devices communicated smoothly without any interruptions. After setting up SSH, the next step involves assigning static IP addresses to each device in the cluster. Static IPs prevent inconsistencies in network communication and ensure that the nodes can reliably locate and interact with each other. In our setup:

- The master node, responsible for managing the cluster, coordinating jobs, and tracking metadata, is assigned 192.168.10.120.
- The two data nodes, responsible for data stor- age and processing, are assigned 192.168.10.130 and 192.168.10.140, respectively.

By structuring the network in this way, we ensure that the master node efficiently distributes workloads across the avail- able data nodes, enhancing parallel processing capabilities. Additionally, static IP assignment helps prevent connection failures that could arise due to dynamically assigned addresses, ensuring a stable and robust environment for big data process- ing. Once the network is fully established and communication is successfully tested using tools like ping and SSH login checks, we proceed to configure core Hadoop components. From Figure 3, we can see that the IP address is being configured in the host's file. This has been done in all the Datanodes and the Master Node. We can give alias names to our IP addresses as shown in Figure 3.

```
 GNU nano 7.2                    /etc/hosts *
192.168.10.120 master
192.168.10.130 node1
192.168.10.140 node3
```

Fig. 3. Configuring IP addresses

The network configuration for Hadoop Spark cluster in an abstract level is presented in Figure 4. For performing the distributed processing, we formed a network using Star topology which has a great impact in the Hadoop Spark cluster. Whenever any node wants to transmit data to another node, it first transmits data to the central node (switch), and then it transfers the data to all the nodes on the network.

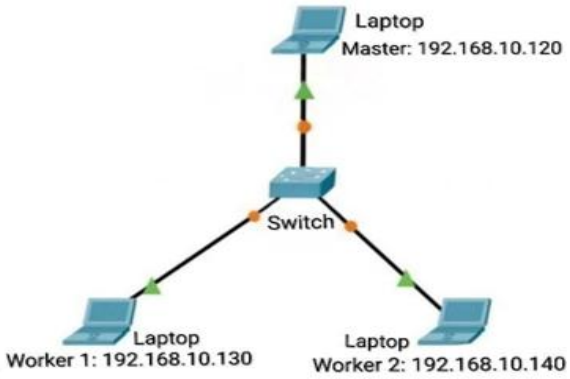

Fig. 4. Network topology

B. Configuration of the Hadoop and Spark

For necessitating the Hadoop multimode cluster effectively, the configuration of the Namenode and Data nodes plays a pivotal role. The Name Node acts as the master node, man- aging the metadata and directory structure of the Hadoop Distributed File System (HDFS), while the Data Nodes are responsible for storing and processing the actual data. Proper configuration ensures seamless communication between nodes, efficient data distribution, and fault tolerance, making the cluster robust and scalable. To enable a Hadoop Multin- ode cluster, it is necessary to make appropriate configura- tions in files such as core-site.xml, hdfs-site.xml, mapred-site.xml, and yarn-site.xml.

C. Initializing and launching the Hadoop-Spark Cluster

After successfully configuring the Hadoop cluster, we pro- ceeded with starting the cluster to ensure all components were functioning correctly. We first launched the Name Node, which manages metadata and coordinates data distribution across the cluster. Using certain commands, we ensured that the data nodes have also been successfully launched and was functioning properly. We used the following com- mands to successfully start all the required daemons of Hadoop and Spark: start-all.sh, start-master.sh and start-workers.sh from the master node. To verify that the cluster was running as expected, we used commands to check the status of HDFS and YARN, ensuring proper communication between nodes. Once the cluster was fully operational, we tested its functionality by running sample Hadoop jobs, confirming that data was being processed effi- ciently across the nodes. This step marked the transition from configuration to execution, allowing us to utilize the cluster for large-scale data processing tasks. To verify that all components were running correctly, we used the jps command to check active processes on each node. Addition- ally, we ran the hdfs dfsadmin -report command to confirm that all DataNodes are successfully connected to the cluster. With the cluster fully operational, we conducted a test run using a sample Hadoop job. This confirmed that data was being correctly distributed, processed, and retrieved across the nodes, validating the effectiveness of our multi- node setup. At this point, the cluster was ready to handle large-scale data processing tasks efficiently. We can access HDFS User Interface (UI), by using the IP address of the Master node (master:9870) to regularly monitor the status of the datanodes and the data distribution in the clusters. YARN (Yet Another Resource Negotiator) plays a pivotal role for management and allocation of resources that are assigned to datanodes. We can access YARN User Interface (UI), by using the IP address of the Master node (master:8088) to regularly monitor the cluster performance like the applications submitted, number of active nodes, un- healthy nodes, etc as shown in the Figure 5.

After executing the tasks, we observed that one cluster node (node1) was marked ‘unhealthy’ due to log directory disk space exceeding 90%, as shown in the above image Figure 5. YARN would subsequently exclude this node from Spark job scheduling for the future tasks, reducing the effective cluster size for our experiments. This operational event contributed to less parallelism than expected and demonstrates the impor- tance of monitoring node health and disk usage in distributed environments.

D. System Configuration Used for the Cluster

- Name Node - Intel i5, 9th Gen, 8 GB RAM, Ubuntu 24.04
- Datanodes - RYZEN 5, 4000 SERIES, 8GB RAM, Ubuntu 24.04
- Datanodes - RYZEN 5, 5000 SERIES, 8GB RAM, Ubuntu 24.04

E. Data Acquisition and Storage

The raw text dataset (Movie reviews) was uploaded to the Hadoop Distributed File System (HDFS). Data was distributed into blocks and replicated on multiple DataNodes, leveraging HDFS's fault tolerance and parallel I/O capabilities. The log- ical file structure and block location metadata were managed by the HDFS NameNode present in the Master Node. We took two different datasets as shown in the below Table II, and performed the analysis by following all the steps, right from uploading the dataset to the HDFS to performing sentiment analysis. The difference between the two data sets is in the file size, one is a smaller dataset compared to the other. The number of rows in the IMDB Movie Review dataset are 176898 and the number of rows in the second dataset user reviews are 61794610. The decision of whether the dataset is smaller or larger is with respect to the Cluster Setup and the availability of the resources.

Table II Dataset details

S.No	Dataset	File Size	Source
1	IMDB Movie Review	300 MB	Dataset Link
2	User Reviews - Movies	12.6 GB	Dataset Link

F. Job Submission and Resource Management

Sentiment analysis tasks were executed interactively through the spark-shell using the YARN cluster manager. Upon job submission, the Spark driver established a SparkCon- text and requested computational resources from YARN's Resource-anager. Application Master would be initiated which negotiates the resources from YARN, and sends the in-structions on the resources and capacity to the Node Managers, which in turn, assign containers on available worker nodes and manage them. Each Spark executor, operating within a YARN container, accessed input partitions from HDFS Data Nodes in parallel.

G. Pseudo-code for Sentiment Analysis

To perform the Sentiment Analysis on the Movie Review for both the datasets, we have followed the algorithm as shown in Algorithm 6. The major steps followed in creat- ing the pipeline are, preprocessing the dataset, by removing unnecessary columns, filling missing data, removing few un- wanted links and text using regular expressions, segregating

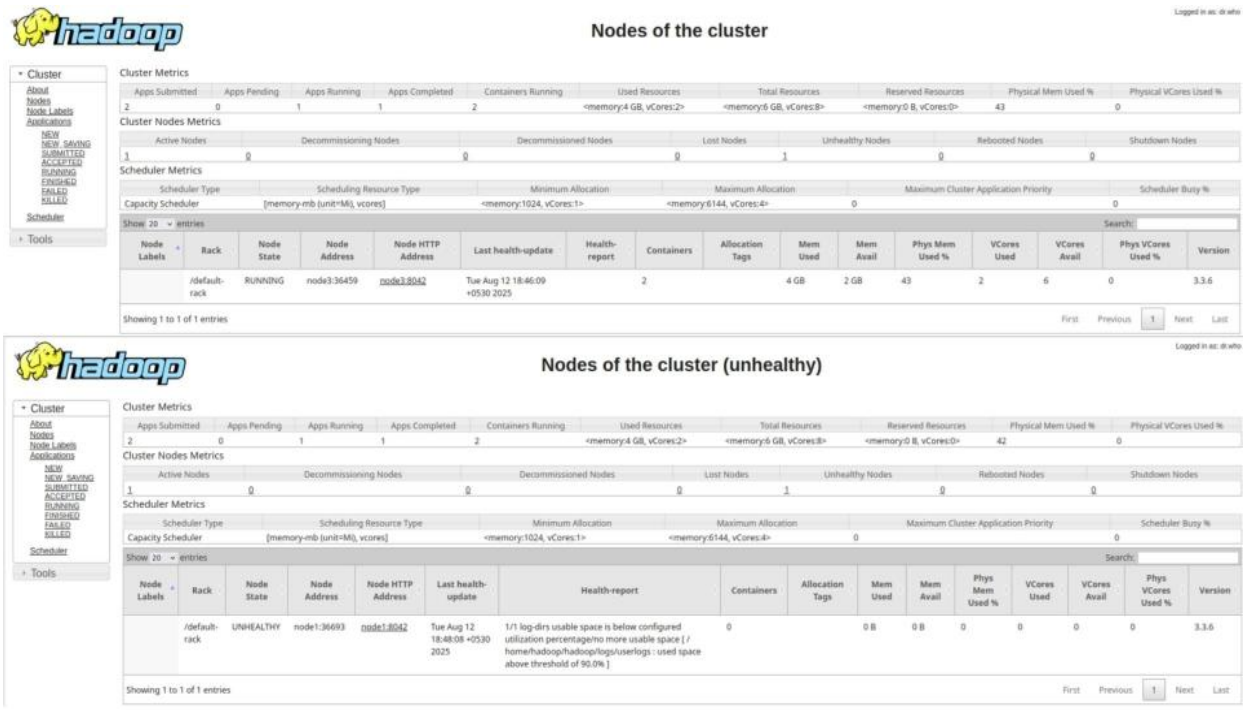

Fig. 5. Yet Another Resource Negotiator (YARN) - Status of the Datanodes

the reviews into sentiments. Following the pre-processing, the dataset is split into training and testing datasets and then we have used Logistic Regression to classify the sentiments as either positive or Negative. Then we can predict the sentiment for a new review. All the below steps were done using Spark's actions and Transformations and by using the Scala Programming Language (Scala API). After performing all the steps in the given pseudo-code, by creating the pipeline, training with the dataset and testing it with a new review, we achieved the output with an accuracy of 76% in classifying the sentiment for the given review.

V. RESULTS AND PERFORMANCE ANALYSIS

A. Execution Time Comparison Across Cluster Configurations

We conducted experiments on a Hadoop-Spark cluster to evaluate performance across different dataset sizes 300 MB and 12.6 GB by comparing execution times for three con- figurations: a single-node cluster, a multi-node cluster with 1 DataNode, and a multi-node cluster with 2 DataNodes. For the 300 MB dataset (Movie Reviews – Sentiment Anal- ysis), the single-node cluster delivered the best performance, outperforming both the 1-DataNode and 2-DataNode setups. This result is consistent with our earlier discussion small datasets suffer from distributed system overheads such as network communication and shuffle costs, job scheduling delays, and resource allocation overhead. Since the single-node setup processes all data locally, it avoids these costs, leading to faster execution.

In contrast, for the 12.6 GB dataset, the multi-node clus- ter with 2 DataNodes significantly outperformed the single- node cluster. Here, the benefits of parallel processing and distributed storage outweighed the fixed overheads of a cluster environment. The higher data volume leveraged the horizontal scalability of the multi-node cluster, resulting in much lower execution times compared to the single-node setup. Addi- tionally, our multi-node configuration provided fault tolerance advantages with a replication factor of 2, data can be re- covered in the event of node failure. While the single- node cluster eliminates distributed overheads, its vertical scal- ing limitations mean it cannot match a well-configured multi- node cluster when handling large-scale data processing.

In Figure 7, we can observe that the execution time taken by the Single Node cluster is comparatively lesser than Multi Node clusters with 1 Datanode and 2 Datanode. While per- forming sentiment analysis on the 300 MB Dataset, Single node cluster was 30% and 40% more efficient than Multi Node- 2 Data nodes and Multi Node - 1 Datanode respectively.

Whereas from Figure 8, we can observe that the multi-node cluster with 2 Data nodes outperformed the Single Node cluster as well as the multi-node cluster with 1 Datanode. Here, while performing Sentiment analysis on 12.6 GB Dataset, the Multi Node cluster with 2 Datanodes was 30% more efficient than the Single Node. Multi Node cluster with 1 Datanode was 17% more efficient than the Single Node. Multi-Node cluster with 2 Datanodes is 15% more efficient than multi-node cluster with 1 Datanode.

B. Impact of File Size on Processing Time

Analyzing execution time against input file size reaffirmed that cluster overhead impacts are inversely proportional to dataset scale. Small to medium-sized files (around 300 MB) presented inefficiencies in cluster mode, with distributed overheads dominating. However, from multiple data points observed, as dataset size increased, performance gains due

Algorithm 1 Sentiment Analysis using Spark for Movie Reviews

Input: MovieReviewCSV
Output: SentimentPrediction

Begin
Start the Spark-shell using the commands mentioned in the Table 2, which launches spark-shell with Scala Programming
Start the Spark session with an app name (e.g. "MovieQuoteSentiment")
Record the start time

Step 1: Load Data
Read CSV file from the HDFS into DataFrame *df*
Select required columns: quote and rating (cast rating as double) in *df*
Filter out rows where quote is null

Step 2: Label Data Columns
For each row in *df*, create Sentiment column: if rating $\geq$ 3.5 then 'Positive', else 'Negative'
Remove rating column

Step 3: Data Pre-Processing - Text Cleaning
Clean quote by removing emojis, tags, and extra spaces using regular expressions
If cleaned quote is null, replace with empty string
Tokenize cleaned quote into words
Remove stop words from tokenized words

Step 4: Define ML Pipeline
Create feature vectors from words (500 features)
Apply IDF transformation to feature vectors
Index 'Sentiment' as numerical label
Set up a Logistic Regression model (maximum Iterations as 10)
Build ML pipeline with all stages above

Step 5: Train/Test Split
Split data into training (80%) and test (20%) sets

Step 6: Train Model
Fit pipeline model to training data

Step 7: Evaluate
Use model to predict sentiment on test data
Evaluate accuracy of predictions
Print accuracy score and the predicted sentiment value
Record end time, calculate the execution time (end time - start time) and convert it into seconds and print the execution time.

Step 8: Predict Single Quote
Create a sample DataFrame with a review (e.g. "this was a good storyline")
Clean the sample quote (remove emoji, tags, extra space; handle null)
Predict sentiment for the sample with the trained model
Convert prediction to label (e.g. Positive/Negative)
Print the quote, cleaned quote, predicted label, and probability

End

Fig. 6. Algorithm: Full pseudocode available at

Sentiment Analysis Task (this link.

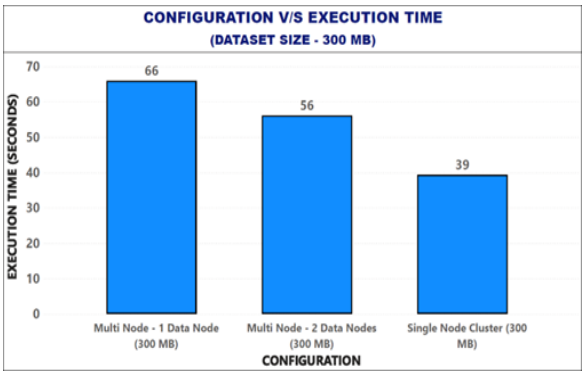

Fig. 7. Configuration vs Execution Time for dataset of size 300 MB

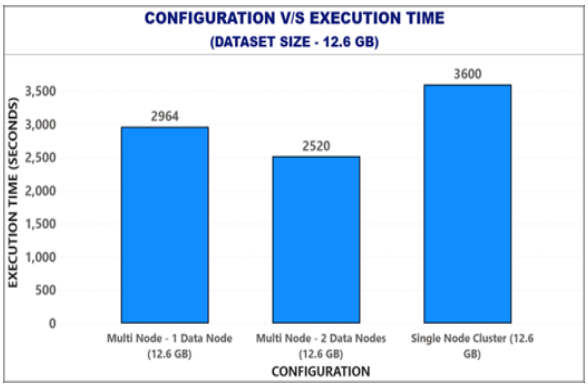

Fig. 8. Configuration vs Execution Time for dataset of size 12.6 GB

to parallel processing improved steadily. This confirms that distributed clusters like ours are best suited and indeed designed for very large-scale data processing.

C. Fault Tolerance and Cluster Resilience

A controlled failure test was conducted wherein a DataNode was intentionally taken offline mid-job. Although the cluster raised an error, the NameNode leveraged replicated blocks stored on the surviving DataNode to continue job execution seamlessly. This behavior underscores Hadoop's robust fault tolerance, ensuring data durability and uninterrupted computation despite node failures, a critical feature for production- grade systems.

D. Distribution of Data

In this project, the data was divided into 128 MB blocks by HDFS and evenly distributed across the two DataNodes in the cluster. Observations confirmed that the data blocks were evenly distributed across available Datanodes with a replication factor of 2. This block-level parallelism is essential in supporting simultaneous data access by multiple compute nodes, reducing bottlenecks and improving throughput for large datasets. This balanced distribution allowed parallel pro- cessing of different data blocks, improving overall throughput and resource utilization. The replication of blocks ensured data redundancy, contributing to fault tolerance during node failures. Efficient data distribution was a key factor in achieving faster processing times on large datasets in the multi-node setup.

After the execution of sentiment analysis using cluster, we got the results much faster than conventional machines and we were able to interpret that the data has been distributed in blocks of size 128 MB in each datanode. From Figure 9, we can see that a certain amount of space has been occupied in each datanode. For our given configuration (Executor Memory, executor cores, etc.) we have achieved the above results. But if we change (increase/decrease) the parameters accordingly the results would vary. The above constraints were fixed based on the available device's capacity.

From the image Figure 10, we can see that the dataset of user_review.csv (size of 12.6 GB), has been divided into 100 Blocks, with each block size as 128 MB and has been stored in both the nodes (node3 and node1 - we can see in the running tasks using spark shell. This interface can give the user regarding the status of the datanode (i.e whether the datanode is alive).

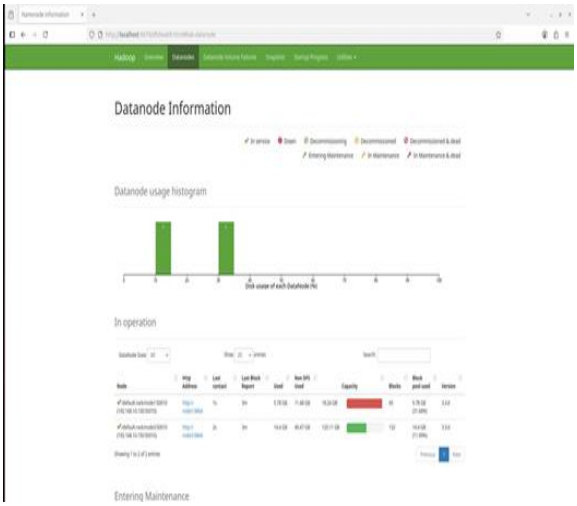

Fig. 9. Data nodes Information

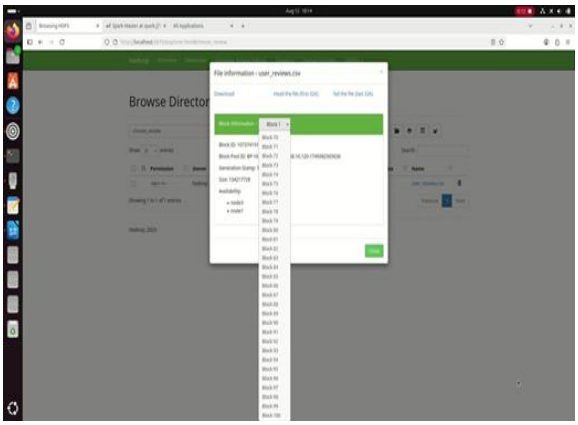

Fig. 10. Data Distribution into blocks

availability). Since the replication factor has been set to 2, the data has been stored in both the datanodes. This image shows the data distribution of a big data split into several blocks. Similarly, for the IMDB Movie Review dataset (size of 300 MB), has been split into 3 blocks and was replicated in both the data nodes.

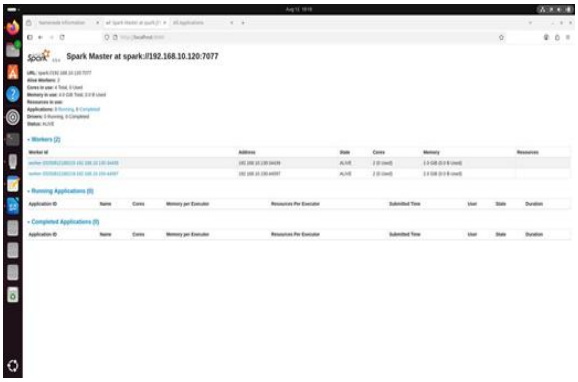

Fig. 11. Spark Interface for viewing running applications and Datanodes

From Spark interface for viewing running applications and Data nodes, we are able to interpret that the spark interface is useful for monitoring the datanodes that are avail- able for the Multinode cluster. This is possible by integrating the Spark with Hadoop with the help of YARN (Yet Another Resource Negotiator). Moreover, we can also view the current

VI. CONCLUSION

In this project, a multi-node Hadoop–Spark cluster was successfully designed, configured, and deployed to address the demands of large-scale data processing. The integra- tion of Hadoop's HDFS with both MapReduce and Apache Spark enabled significant efficiency gains for computation- ally intensive tasks, such as sentiment analysis on extensive text datasets. Incorporating Apache Spark into the cluster brought about substantial improvements in processing speed through its in-memory computation, thus reducing latency and making iterative analytics more efficient. The results consistently demonstrated that multi-node deployments not only achieved reduced execution times and enhanced scal- ability, but also ensured robust fault tolerance and resource optimization. These outcomes underscore the effectiveness of distributed computing architectures for contemporary big data applications and highlight the practical benefits of combining Hadoop and Spark in clustered environments. Through this study, we not only explored the technical aspects of setting up and configuring a Hadoop–Spark cluster but also gained valuable insights into how distributed computing can transform the way we handle big data, contributing to the growing body of research that demonstrates the practical benefits of these technologies in real-world scenarios.

VII. FUTURE SCOPE

While the current Hadoop–Spark cluster setup has proven robust for batch-oriented sentiment analysis, extending the cluster to cloud-based or hybrid environments can facilitate dynamic scaling and more flexible resource provisioning. The integration of Spark Streaming can further empower the platform to handle real-time data ingestion and anal- ysis, supporting tasks such as live sentiment tracking and anomaly detection. Employing advanced analytics through Spark MLlib, as well as integrating deep learning frameworks with the cluster, could expand analytical capabilities from basic sentiment classification to more sophisticated predictive modeling and natural language processing applications.

ACKNOWLEDGMENT

We would like to extend our sincere gratitude to our guide, Dr. A Nagaraja Rao, for providing the invaluable opportunity to start this project. His continuous mentorship, detailed guidance, and support in providing the necessary tools to build the Hadoop-Spark cluster were instrumental in shaping this paper.

We also wish to express our appreciation to the faculty of the School of Computer Science and Engineering (SCOPE) at VIT, Vellore. We are particularly grateful for their careful review and insightful feedback during two project reviews, which greatly helped in refining our methodology and results.

Finally, we are grateful to the anonymous reviewers for their comments, which helped improve the quality of this manuscript.

REFERENCES

[1] O. Haddad, F. Fkih, and M. N. Omri, "An intelligent sentiment prediction approach in social networks based on batch and streaming big data analytics using deep learning," *Soc. Netw. Anal. Min.*, 2024.

[2] Apache Software Foundation, "Apache Hadoop documentation." [Online]. Available: https://apache.github.io/hadoop/hadoop-project-dist/hadoop-common/ClusterSetup.html

[3] Apache Software Foundation, "Apache Spark documentation." [Online]. Available: https://spark.apache.org/docs/latest/running-on-yarn.html

[4] V. Vijay, V. Sharma, V. Srivastava, and V. K. Jain, "A comparative study on Hadoop MapReduce and Apache Spark framework for big data analytics," *Int. J. Res. Publ. Rev.*, vol. 5, no. 2, pp. 3228–3232, 2024.

[5] P. Shah and N. Malu, "An Apache Spark implementation for sentiment analysis on Twitter data," *Int. Res. J. Mod. Eng. Technol. Sci.*, vol. 6, no. 3, pp. 1857–1861, 2024.

[6] M. Bergui, S. Hourri, S. Najah, Z. Zhang, *et al.*, "Predictive modelling of MapReduce job performance in cloud environments using machine learning techniques," *J. Big Data*, vol. 11, p. 98, 2024.

[7] J. Dean and S. Ghemawat, "MapReduce: Simplified data processing on large clusters," *Commun. ACM*, vol. 51, no. 1, pp. 107–113, 2008.

[8] M. Sokolova, "Big text advantages and challenges: Classification perspective," *Int. J. Data Sci. Anal.*, vol. 5, no. 1, 2018.

[9] S. Tang, B. He, C. Yu, *et al.*, "A survey on Spark ecosystem: Big data processing infrastructure, machine learning, and applications," *IEEE Trans. Knowl. Data Eng.*, vol. 34, no. 1, pp. 71–91, 2022.

[10] K. Sandhu, "Big data with cloud computing: Discussions and challenges," *Big Data Min. Anal.*, vol. 5, no. 1, pp. 32–40, 2022.

[11] Zarei, S. Safari, M. Ahmadi, *et al.*, "Past, present and future of Hadoop: A survey," *arXiv preprint arXiv:2202.13293*, 2022.

[12] Ullah, S. Dhingra, X. Xia, and M. A. Babar, "Evaluation of distributed data processing frameworks in hybrid clouds," *J. Netw. Comput. Appl.*, vol. 224, p. 103837, 2024.

[13] L. Owusu-Berko, "Harnessing big data, machine learning, and sentiment analysis to optimize customer engagement, loyalty, and market positioning," *Int. J. Comput. Appl. Technol. Res.*, vol. 14, 2025.

[14] S. Lim and D. Park, "Improving Hadoop MapReduce performance on heterogeneous single board computer clusters," 2024.

[15] S. Bhathal and A. Singh, "Big data: Hadoop framework vulnerabilities, security issues and attacks," 2019.

[16] K. Kalia and S. Dixit, "Improving MapReduce heterogeneous performance using KNN fair share scheduling," 2022.

[17] M. R. Sundara Kumar and H. S. Mohan, "Improving big data analytics data processing speed through MapReduce scheduling and replica placement with HDFS using genetic optimization techniques," 2024.

[18] M. R. Sundarakumar, R. Sharma, and S. K. Fathima, "Improving data processing speed on large datasets in a Hadoop multi-node cluster using enhanced Apriori algorithm," 2023.

[19] J. Xu, Y. Wang, and L. Zhou, "Resource-aware scheduling strategies in heterogeneous Hadoop clusters," *Future Gener. Comput. Syst.*, vol. 145, pp. 203–214, 2023.

[20] Y. Chen, H. Li, and X. Zhang, "Performance comparison of Hadoop MapReduce and Spark in hybrid workloads," *Concurrency Comput. Pract. Exp.*, vol. 35, no. 12, p. e7593, 2023.