Science & Technology Development Journal 2024, 27(3):3496-3506

Open Access Full Text Article

Research Article

A System For Storing And Processing Big Data Based On The Apache Spark Platform

Nguyen Anh Tuan, Vo Hong Phuong, Cao Tien Thanh, Tran Khai Thien*

ABSTRACT

The primary objective of this paper is to investigate and implement the Apache Spark big data processing platform on a stock dataset, followed by the application of a machine learning technique for prediction and modeling. Specifically, PySpark, a Python API for Apache Spark, is utilized to facilitate the interaction. The Spark MLlib library is employed for data transformation, whereas the GraphX library is used for data modeling. Multiple executions of the experimental program demonstrated significant performance improvements, with notably shorter runtimes on the Spark cluster than on a single machine. These results highlight the advantages of distributed and parallel processing in large-scale data analysis.

Key words: Apache Spark, machine learning, big data, prediction model, Spark Mlib, GraphX, PySpark

Ho Chi Minh City University of Foreign Languages and Information Technology, Ho Chi Minh City, Viet Nam

Correspondence

Tran Khai Thien, Ho Chi Minh City University of Foreign Languages and Information Technology, Ho Chi Minh City, Viet Nam

Email: thientk@huflit.edu.vn

History

- Received: Jul 05, 2024
- Revised: Sept 06, 2024
- Accepted: Sept 14, 2024
- Published Online: Sept 30, 2024

DOI :

https://doi.org/10.32508/stdj.v27i3.4419

VNUHCM PRESS

INTRODUCTION

In recent years, terms such as artificial intelligence (AI), machine learning (ML), and big data have become increasingly prominent. Among these, big data has quickly become a key field that underpins various industries and is currently facing a significant shortage of skilled personnel. Forbes reported that between 2016 and 2018, 90% of the world's total data were produced. These datasets are so large that they exceed the capacity of traditional tools and software for visualization, management, and processing, thereby leading to the formal definition of "big data." Managing big data effectively requires advanced approaches and specially designed technologies, as existing tools often lack the capacity to handle such massive data volumes efficiently.

The COVID-19 pandemic from late 2019 through 2021 underscored the need for robust big data solutions, as work environments shifted rapidly to remote, online models that depended on large volumes of data, virtual interactions, and collaborative platforms. This period saw an unprecedented surge in data generation. In this context, the application of big data processing platforms has become crucial for supporting data analysis, providing business forecasting solutions, evaluating remote work efficiency, and addressing medical challenges.

As data generation continues to increase, there is an urgent need for storage and processing platforms capable of delivering rapid query responses. With the current scale of big data, speed has become a competitive advantage. Commonly used tools such as Microsoft Excel are inadequate for large-scale data management; for instance, Excel's column limitations and its reliance on a single-machine's memory restrict its capacity with large datasets. Instead, platforms capable of horizontal scaling, such as those utilizing clusters of computers to distribute tasks, present a more viable solution for processing and analyzing large datasets. Apache Spark, with its architecture that supports the division and concurrent processing of data across numerous nodes, effectively meets these requirements. Unlike traditional tools, Spark's architecture allows real-time data handling, significantly outperforming even Hadoop-based systems, which often experience delays in processing speed.

Various studies have examined big data applications, such as COVID-19 data analysis, weather forecasting, and stock price prediction [1]. However, many of these studies remain at an introductory level or rely on the Hadoop framework. Our study stands out by employing Apache Spark for predictive analytics on financial data, highlighting Spark's substantial benefits in speed and scalability.

The main contributions of this paper include the following:

- Outlining a structured approach for big data processing through Apache Spark, focusing on efficient data gathering, visualization, preprocessing, and application of machine learning to stock data via PySpark.

Cite this article : Tuan N A, Phuong V H, Thanh C T, Thien T K. **A System For Storing And Processing Big Data Based On The Apache Spark Platform**. *Sci. Tech. Dev. J.* 2024; 27(3):3496-3506.

- Using distributed workers in the processing and execution stages significantly shortens the computation time, enabling more experimental runs and enhancing model selection through evaluation metrics.

The remainder of the paper is organized as follows: Section II presents the theoretical background. Section III introduces the problem and the proposed solution. The experimental results are presented in Section IV. Finally, the conclusion is provided in Section V.

BACKGROUND

A. MapReduce model

First, MapReduce is a well-known distributed computing model developed by Google and introduced in 2004[2]. This model works by dividing a task into smaller subtasks, which are executed in parallel, and then combining the results to produce the final output. The MapReduce process can be summarized as follows: reading the input data, processing the input via the map function, sorting and merging the results from distributed systems, aggregating the intermediate results via the reduce function, and generating the final result (see Figure 1).

B. Apache Hadoop

Hadoop is an open-source platform, developed in Java, designed for the storage and processing of big data and implementing Google's MapReduce model[3]. The core structure of Hadoop comprises four key components: Hadoop Common, a necessary library for the other modules; Hadoop YARN, a framework responsible for managing cluster processes and resources; Hadoop Distributed File System (HDFS), a distributed file system that automatically splits files into smaller blocks, replicates them, and stores them across multiple servers to ensure fault tolerance and high availability; and Hadoop MapReduce, which operates on YARN to facilitate the parallel processing of large datasets.

C. Apache Spark

Apache Spark[4] is an open-source cluster computing platform designed for real-time data processing, with a key feature being its in-memory cluster computing, which significantly increases the application processing speed. Spark offers an interface for parallel data processing while enhancing fault tolerance. It is capable of handling various workloads, including batch processing, iterative algorithms, interactive queries, and streaming.

1) The key features of Apache Spark include the following:

· Speed: Spark processes data up to 100 times faster than does Hadoop MapReduce.

· In-Memory Caching: This method accelerates computations through in-memory caching.

· Scalability: Spark scales efficiently via Cluster Manager, YARN, or Hadoop.

· Real-Time Processing: This method supports low-latency, real-time data processing.

· Multilanguage Support: Spark APIs are compatible with Java, Scala, Python, and R, with Scala as the default language.

2) The Spark architecture is based on two essential components: resilient distributed datasets (RDDs) and directed cyclic graphs (DAGs). It features a layered architecture and integrates various libraries and extensions.

3) Operating mechanism:

· Distributed Computing: Spark partitions and distributes data tasks across multiple machines in a cluster, enabling the processing of large datasets.

· In-Memory Computing: Unlike traditional systems that store data on hard drives, Spark stores data directly in the RAM, significantly increasing processing speed.

· Support for Various Data Processing Types: Spark handles batch processing, real-time data (streaming), interactive analytics, and other data forms.

· Multi-Language APIs: This provides APIs for widely used programming languages, including Scala, Java, Python, and R, making it developer friendly.

· Graph Processing and Machine Learning: Spark includes tools such as GraphX for graph processing and MLlib for machine learning, extending its analytical capabilities.

Architecture and Data Processing Methods

In the architecture of Apache Spark (Figure 2), the **driver program** plays a crucial role in managing and controlling the entire data processing workflow across a cluster of computers. The driver program initiates **SparkContext**, which contains the necessary functions for performing data processing tasks.

The **Spark Driver**, an essential part of the Driver Program, operates on a computer within the cluster and oversees the data processing activities across the cluster. It works in tandem with the **cluster manager**, a resource manager that allocates and monitors tasks among the computers (nodes) in the cluster to ensure efficient task execution.

One of the key components in Spark's architecture is the **RDD**, a specialized data structure that is created within SparkContext. Once created, the RDD is

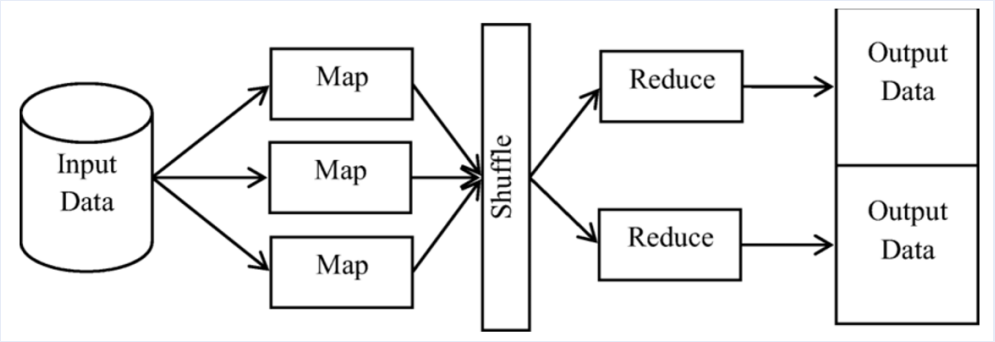

Figure 1: MapReduce model

distributed and stored across multiple worker nodes in the cluster. These **worker nodes** execute tasks assigned by the Cluster Manager and subsequently return the processed results to SparkContext.

Additionally, executors run on various computers within the cluster, which are managed by the Spark Driver. Executors are responsible for performing specific data processing tasks, and each executor can handle multiple tasks concurrently, contributing to the overall parallel processing capability of Spark.

D. PySpark

PySpark[6] is an API for Apache Spark that enables the use of Spark's functionalities through the Python programming language. It serves as a powerful tool for handling big data or real-time data processing tasks. A key distinction of PySpark is its provision of the PySpark shell, which allows users to directly interact with and analyze data within a shell environment. This feature enhances PySpark's ability for fast, robust data processing and analysis, maximizing its potential for various data-intensive tasks. By combining Python's flexibility with Apache Spark's distributed computing power, PySpark is capable of efficiently processing and analyzing datasets of varying sizes, catering to a wide range of user needs.

Additionally, PySpark offers several notable features and benefits:

- Multisource data processing: PySpark supports data from diverse sources, including HDFS, Apache HBase, Apache Cassandra, JSON, CSV, Parquet, Avro, and other formats, facilitating the handling of multisource data.
- High performance: This method is designed for high-speed performance, leveraging optimizations and distributed computing, with an in-memory data processing model that minimizes the I/O time and enhances the processing speed.
- Integration: PySpark seamlessly integrates with other Big Data tools, such as Apache Hadoop, Apache Hive, Apache HBase, and Apache Kafka, allowing flexible incorporation into existing systems.
- Automatic resource management: Through Apache YARN or Apache Mesos, PySpark can manage resources automatically, adjusting the number of tasks on the basis of application demands to optimize resource utilization and ensure efficient performance.
- Machine learning support: PySpark includes a variety of machine learning algorithms via the MLlib library, enabling easy and effective development and deployment of machine learning models on large datasets.

E. Some related work

Extensive research has leveraged big data platforms, particularly Apache Spark, for prediction and recommendation systems because of their scalability and efficiency in handling large datasets. In an early study, [[7] employed big data tools such as Apache Spark for photovoltaic forecasting, applying algorithms such as linear support vector machines. This approach demonstrated significant improvements in both processing efficiency and prediction accuracy, laying the groundwork for subsequent research.[8] advanced this research by utilizing Apache Spark and MLlib for stock classification and volatility prediction. They compared several algorithms, including naïve Bayes, random forest, decision tree, and logistic regression. Their experiments indicated that the random forest and decision tree methods consistently provided superior accuracy, with higher AUC and PR values, confirming their scalability. Subsequent studies, such as[9], who introduced more advanced techniques, including long short-term memory (LSTM)

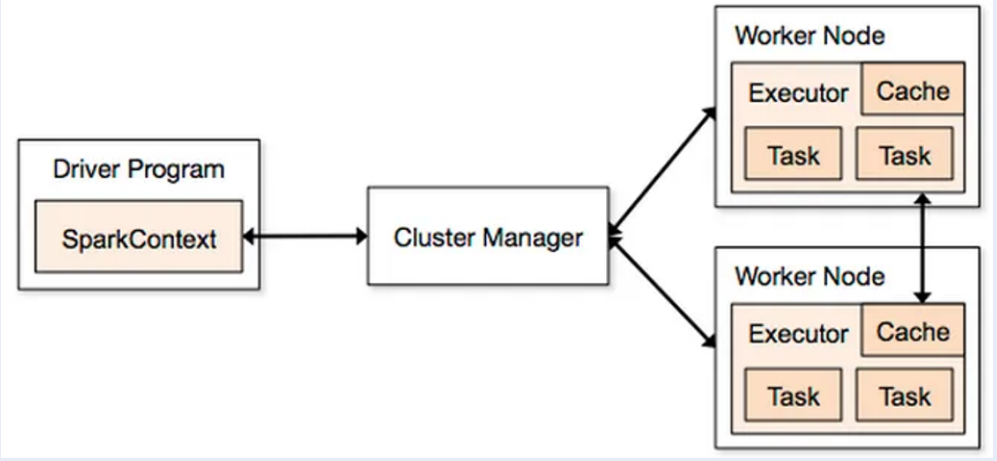

Figure 2: Architecture and Data Processing Methods [5]

neural networks, demonstrated that LSTM models outperform traditional approaches. [10] explored real-time stock trend prediction via Apache Spark streaming. By employing machine learning algorithms such as SVM, their system achieved notable improvements in prediction speed and accuracy, making it particularly suitable for high-frequency trading scenarios.

Moreover, beyond the financial sector, Apache Spark has also been applied to recommendation systems. Ha and Nguyen [11] utilized Spark and its MLlib library to implement matrix factorization techniques for movie rating predictions and recommendation systems, respectively. This study employed the alternating least squares (ALS) algorithm for collaborative filtering, which achieves high accuracy by optimizing the ALS parameters for large datasets. Recent work has applied Apache Spark to real-time data forecasting in IoT networks, demonstrating its effectiveness in time series analysis for streaming data [12]. Another study combined a deep LSTM model with the Jaya anti-coronavirus optimization (JACO) algorithm on Spark, which yielded improved accuracy in stock market prediction [13].

This paper builds upon previous studies by utilizing Apache Spark to process two distinct datasets: one consisting of stock transaction data from 15 technology corporations and another from Sparkify, a virtual music streaming service. By integrating linear regression with PySpark, this study not only effectively predicts financial data but also highlights Apache Spark's ability to handle large, diverse datasets with significant differences in size and structure. The use of distributed workers further highlights Spark's scalability and efficiency for large-scale data analysis.

PROBLEM ANALYSIS

A. Requirements

Predictive modeling presents a significant challenge across various fields, including finance, healthcare, weather forecasting, and environmental studies. Machine learning models have emerged as a popular tool for tackling these challenges, particularly in the domain of stock price prediction, which involves leveraging historical data and analytical methods.

Linear regression is one of the most widely used machine learning models for stock price prediction. This model operates on the assumption of a linear relationship between input variables and stock prices. By incorporating input features such as financial indicators, economic factors, and market data, linear regression enables the prediction of stock prices on the basis of the linear correlation between these variables and stock movements.

When machine learning models are combined with techniques from both technical and fundamental analyses, they offer investors and financial analysts a more holistic and accurate perspective on stock market trends and the outlook for individual stocks.

B. Technology Selection

Selecting the appropriate technology for building a stock price prediction model is crucial for optimizing performance and flexibility. In evaluating different development environments, Jupyter Notebook has emerged as a suitable option because of its strong alignment with data science workflows and compatibility with big data tools, particularly Apache Spark. This advantage is less prominent in other tools, such

as RStudio or PyCharm. Jupyter Notebook is a suitable choice for developing a foundational platform. Its user-friendly interface supports gradual project exploration and allows for easy expansion in the future. However, for extremely large datasets, Jupyter may encounter performance limits, making it more suited to exploratory analysis rather than high-volume processing tasks.

The notebook format, which integrates code with narrative text, offers clear benefits for both coding and documentation. Jupyter Notebook's environment encourages iterative coding, where immediate feedback can guide adjustments and streamline the debugging process. Additionally, errors are isolated within the notebook itself, enabling quick navigation back to prior steps without affecting the main server. Its structure also facilitates collaborative work and documentation, making it an accessible choice for team-based data science projects.

C. Machine Learning Algorithms

Machine learning significantly advances data science and artificial intelligence by creating systems capable of autonomous learning and adaptation without detailed programming. These systems generate predictive models via input data and accumulated experience. Key approaches include supervised learning, which predicts outputs on the basis of labeled data; unsupervised learning, which handles tasks such as clustering without labels; semisupervised learning, a combination of both; reinforcement learning, where the system learns from outcomes without explicit instructions; and deep learning, which mimics human brain functions via neural networks. Effective machine learning models rely on proper data preprocessing and model selection.

This research uses the linear regression model[14], which assumes a linear relationship between variables to predict outcomes on the basis of input data. Linear regression is widely used in various domains because of its simplicity and interpretability, helping businesses transform data into insights, and scientists analyze trends. It also serves as a foundational tool in more complex machine learning tasks.

EXPERIMENTAL RESULTS

A. Datasets

The experimental project uses two datasets to evaluate Apache Spark's performance:

- The first dataset consists of daily stock transaction data from 1972–2024, covering 15 major technology companies. It was obtained via the YFinance Python library, which collects financial data from Yahoo Finance. Despite its size of 12.2 MB, the dataset contains more than 100,000 records, which fulfills the volume criteria typically associated with big data. Additionally, the continuous flow of stock market data reflects the velocity component, where timely processing is crucial for decision-making. Although relatively small in storage size, the complexity and structure of the data—along with the need for quick processing—make it appropriate for Big Data analysis in this context. The dataset also displays variety, incorporating different data types such as timestamps, prices, and company information, necessitating sophisticated data handling techniques.
- The second dataset comes from Sparkify, a virtual music streaming platform. It includes over 26 million user interaction records, resulting in a dataset of 12 GB. This large volume alone makes it a clear example of Big Data. Furthermore, the platform records user actions in real time, such as songplay and navigation through the service, which aligns with the velocity aspect of big data. The dataset is also diverse, containing both structured information such as session IDs and song titles, as well as more complex semistructured logs of user behavior. Given these characteristics, Apache Spark is leveraged to efficiently process and analyze these data in a distributed manner.

B. Configuration setup and measurement method

- Parameter settings: Spark is configured to use 4 cores (CPU) and 4 GB of RAM per executor.
- Total workers: The setup includes 4 workers, each with 4 cores (CPU) and 4 GB of RAM.
- Resource allocation: In total, Spark will utilize up to 16 cores (CPU) and 16 GB of RAM across all executors, which are distributed among the 4 workers (4 cores and 4 GB per worker).
- Measurement formula: The execution time for each cell in Jupyter Notebook is measured via the "time" module in Python. The command "%%time" is placed at the beginning of each cell to display the runtime once the query is completed.
- Measurement method: Two measurement scenarios are used for comparison:

+ Execution using 1 worker.
+ Execution using 4 workers.

C. Experiments

a) Experiment on the YFinance stock dataset:

The total number of stock trading samples, 103,285, is summarized in Table 1 and Table 2.

Table 1: The collected data

Company name	Quantity
Ford	13080
Pfizer	13080
Intel	11114
Apple	10926
Micron Tech	10049
Microsoft	9600
Amazon	6774
NVIDIA	6349
Google	4948
American Airlines	4669
Tesla	3473
Meta	2996
Alibaba	2409
Riot	2025
Snap	1793

Data Preprocessing

Two columns, Dividends and Stock Splits, were removed because they did not contribute to the analysis. Since both columns contained only zero values, their removal helped eliminate unnecessary noise in the data and reduced the risk of overfitting, which in turn could enhance model performance. Additionally, the date column was excluded to prevent any potential data leakage, ensuring that the model could be generalized effectively when applied to new, unseen datasets.

For categorical columns, such as Company Name and User ID, string data are encoded via one-hot encoding, which converts these categorical variables into numeric vectors that the model can process. This method guarantees that no ordinal relationships are assumed between categories. Numerical data, including stock prices and trading volumes, were vectorized via VectorAssembler to merge multiple feature columns into a single feature vector for input into the model. This approach ensured that both the encoded categorical data and the numerical features were correctly formatted for the machine learning algorithms in Spark.

Model Creation and Training

The dataset is divided into training and test sets via the randomSplit method with ratios of [0.8, 0.2]. This method allocates 80% (82,753 samples) for the training set and 20% (20,532 samples) for the test set. The parameter seed = 42 ensures consistent random splitting across multiple runs if needed.

Feature Vector Preparation and Data Normalization:

- VectorAssembler: Combines feature columns ('Open', 'High', 'Low', 'Volume') into a single vector. This allows the model to analyze all feature data efficiently in predicting future stock prices.
- MinMaxScaler: MinMaxScaler normalizes data to a range (typically 0–1). Given the variability in stock prices and volumes, normalization standardizes features, reduces the influence of value fluctuations, and improves the convergence speed and accuracy of machine learning models.

A pipeline is used to streamline the process by automatically combining the feature columns into a vector and normalizing the data. The pipeline processes both the training and test datasets, ensuring the consistency of the preprocessing steps for model training and evaluation.

Hyperparameter Tuning:

- ParamGridBuilder: Constructs a grid of hyperparameters for grid search. The addGrid method is used to experiment with different model hyperparameters.
- TrainValidationSplit: Trains the model with each hyperparameter combination from the grid on the training set and evaluates the performance on the validation set. The best parameter set is selected on the basis of the evaluation results via the regression evaluator.

The 'Close' column, which represents the closing stock price, is chosen as the target prediction label for both models, as it accurately reflects stock trends and investment performance over time.

Table 2: Overview of the data description

Fields	Description	Data type
Date	Trade date	Timestamp
Open	Open price	Double
High	Highest price of the day	Double
Low	Lowest price of the day	Double
Close	Close price	Double
Volume	Volume (number of shares traded)	Long (int)
Dividends	Dividend amount paid to shareholders	Double
StockSplits	Stock split quantity	Double
Company	Stock name	String

Linear Regression

To identify the optimal set of hyperparameters for the model, **ParamGridBuilder** is used to explore the following parameters:

- **regParam (Regularization Parameter)**: This parameter helps mitigate overfitting in linear regression by applying regularization to the model. It balances minimizing the training error and adding a regularization penalty to the model. A higher regParam increases the regularization strength. The tested values for this parameter are [0.001, 0.01, 1.0].
- **elasticNetParam (Elastic Net Parameter)**: Elastic Net combines the L1 (Lasso) and L2 (ridge) regularization techniques in linear regression. ElasticNetParam controls the ratio between these two methods. When elasticNetParam = 0, only L2 regularization (ridge) is applied, and when elasticNetParam = 1, only L1 regularization (Lasso) is used. Intermediate values between 0 and 1 represent a mix of both regularization types. The tested values for this parameter are [0.0, 0.5, 1.0].

We use two metrics, RMSE (root mean square error) and R-squared, to evaluate the model.

Root mean square error (RMSE) for the test data: The RMSE is 0.68, indicating that the average deviation between the model's predictions and the actual values is relatively low.

R-squared (R^2) for the test data: The R^2 value is 0.99, indicating that the model explains 99% of the variance in the dependent variable, which suggests strong model performance.

Best model parameters:

- regParam: The optimal value for the regularization parameter is 0.001, indicating minimal regularization.
- ElasticNetParam: The optimal value for elasticNetParam is 0.0, meaning that the model exclusively uses L2 regularization (Ridge) without incorporating L1 regularization (Lasso).

The model's prediction chart is as follows

The prediction results for the entire test dataset are presented in Figure 3.

On the basis of the metrics, visualization charts, and actual prediction values, the linear regression model delivers outstanding results and performance on the test set. It demonstrates strong generalization capabilities and effective learning, making it a reliable option for application to real-world data.

Comparison and explanation using 1 worker and 4 workers

On the basis of Figure 4, when performing data processing tasks in machine learning, there is a notable difference in execution time between using 1 worker and 4 workers. Reading data from a file took 24 seconds with 1 worker, compared with 27.3 seconds with 4 workers. This suggests that for smaller datasets, increasing the number of workers does not necessarily improve performance. In fact, with the allocated CPU cores and RAM, the use of multiple workers can lead to slower processing speeds.

For training a linear regression model, the execution time was consistently short, ranging from 1 to 2 s in the experimental trials. Using a Spark cluster with multiple workers did not significantly reduce the data processing or execution time. In fact, 4 workers took

Table 3: The parameters of linear regression

Root Mean Squared Error (RMSE) on test data = 0.68	
R-squared on test data = 0.99	
Best model parameters	
regParam: 0.001	elasticNetParam: 0.0

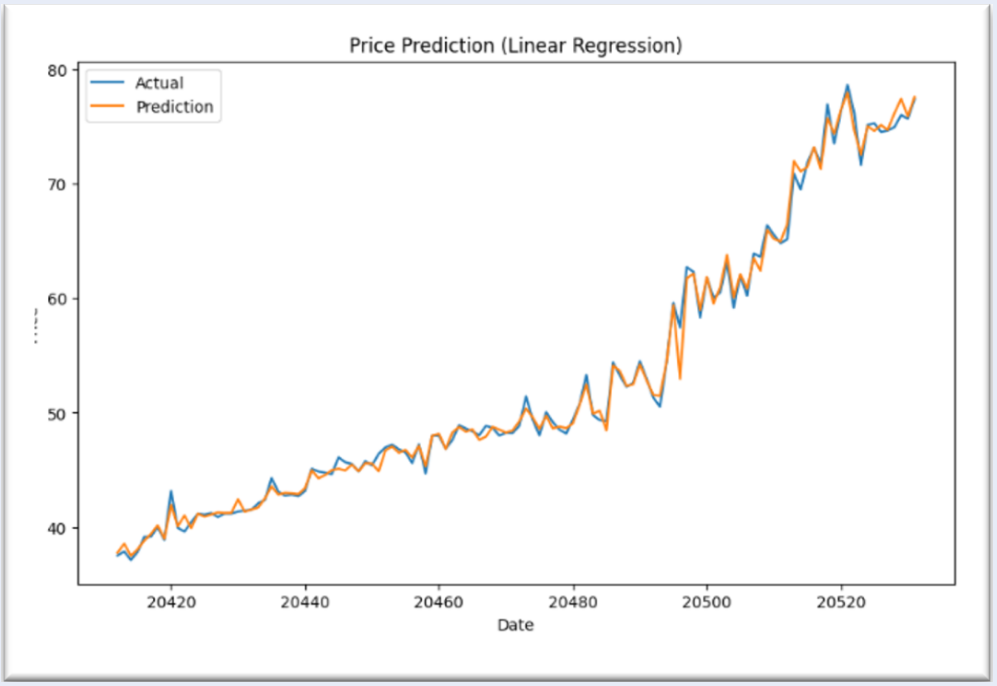

Figure 3: Prediction results on the test set of linear regression

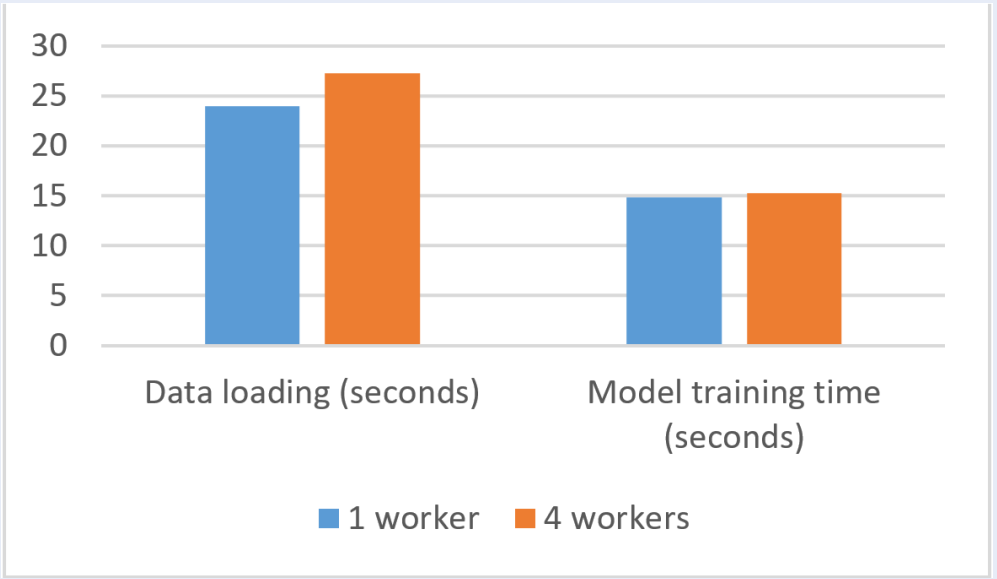

Figure 4: Comparison of execution times when using 1 worker and 4 workers

approximately 1 second longer than 1 worker. This indicates that, for models with such short execution times as linear regression, employing multiple workers might currently lead to reduced efficiency. This outcome highlights the need for further consideration when deciding whether to use multiple workers, especially for tasks with short execution times.

b) Experiment on the Sparkify stock dataset

Sparkify is a simulated music streaming platform, modeled after real-world companies such as Spotify and Pandora. Millions of users engage with music services daily, either through free, ad-supported plans or premium subscriptions with additional features and no ads. Users can upgrade, downgrade, or cancel their subscriptions at any time, making customer satisfaction a key factor. Each user interaction, such as song playback, playlist additions, ratings, friend requests, logins, logouts, and setting changes, generates valuable data. These activity logs provide insights into user satisfaction and engagement with the platform.

The Sparkify dataset contains 26,259,199 rows and 18 feature columns, offering a comprehensive overview of user behavior and interactions. A detailed description of the dataset is presented in Table 4.

The categorical columns, such as Artist, Auth, Gender, and Page, were transformed via one-hot encoding to convert them into numerical vectors for model training. Numeric features such as Length and Item in Session were normalized via MinMaxScaler to ensure that all the values fell within the same range. Finally, VectorAssembler was applied to combine both encoded categorical features and normalized numeric features into a single feature vector, preparing the dataset for machine learning algorithms in Spark.

Comparison and explanation using 1 worker and 4 workers

On the basis of Figure 5, when performing data processing tasks in machine learning with a relatively large dataset (approximately 12 GB), the operations—from reading data from files to handling null data and running training models—exhibit significant differences between 1 worker and 4 workers. Therefore, applying Apache Spark for processing large datasets is entirely feasible.

- Scale-Out:

The Spark cluster has the ability to scale horizontally, utilizing multiple nodes to process data concurrently. This enables faster program completion than running on a single machine.

- Resource utilization:

Cluster Spark effectively utilizes distributed resources by dividing tasks and processing data across nodes. This maximizes the available resources on each node and minimizes the waiting time due to parallel processing.

- Performance Boost:

Using Cluster Spark can enhance the performance of large data processing tasks, especially when simultaneous and parallel processing is essential.

The reduced runtime when Spark is used on a cluster, compared with a single machine, highlights the advantages of distributed and parallel data processing.

Given the experimental results, linear regression was determined to be a suitable model because of its simplicity and efficiency. With an RMSE of 0.68 and an R^2 of 0.99, the model successfully captured 99% of the variance in stock price data, demonstrating strong predictive capabilities. Additionally, linear regression is computationally lightweight and straightforward to implement on distributed systems such as Apache Spark, making it an ideal choice for processing the current dataset without introducing unnecessary complexity or overfitting. For the Sparkify dataset, which includes over 26 million rows of user activity, linear regression also demonstrated its effectiveness. Despite the dataset's size and complexity, linear regression captures the relationships between the key features and delivers reliable predictions. Its simplicity, combined with low computational overhead, makes it particularly appropriate for large datasets such as Sparkify. Furthermore, linear regression integration with Apache Spark enables efficient data processing and model training, making it a practical solution for this application.

CONCLUSION

This paper investigates the application of data training and prediction via the linear regression model, which is supported by the PySpark MLlib library, across two distinct datasets: stock transaction data and user activity data from the Sparkify music streaming platform. The linear regression model demonstrated solid performance in predicting stock prices, achieving consistent accuracy on the basis of metrics such as the RMSE and R-squared. When applied to actual financial data, the model retained its prediction capabilities, showing its effectiveness in analyzing stock trends. For the Sparkify dataset, although

Table 4: Overview of data description

Field	Description	Data type
Artist	Name of the singer with a song played during the event	String
Auth	Description of the user authentication status: login, logout, guest access	String
First Name	Customer's name	String
Gender	Gender	String
Item in Session	Number of events occurring in a user session	Long
Last Name	Customer's last name	String
Length	Duration of the song	Double
Level	User hierarchy: paid or free	Double
Location	User location	String
Page	Page where events occur, such as package cancellation, upgrade, down-grade…	String
Registration	Number of subscriptions per user	Long
Session ID	Session identifier	Long
Song	Song title	String
Status	HTTP response status code (200, 500, 301, 404…)	Long
Timestamp (TS)	Time measured in milliseconds	Long
User Agent	Information about the accessing device	String
User ID	Identifier associated with each user	String
Method	HTTP server call method (POST/GET)	String

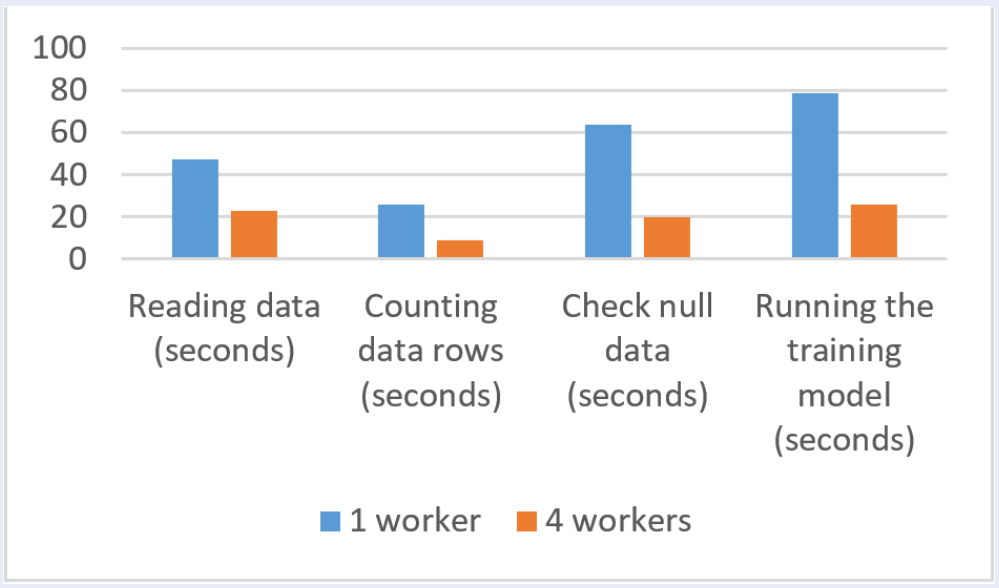

Figure 5: Comparison of the operations: reading, writing, data processing, and training with 1 and 4 workers

the model was primarily aimed at stock data, the experiments demonstrated PySpark's efficiency in processing large-scale user interaction data, easily handling millions of records. The deployment of tasks via multiple executors on a cluster further highlights the benefits of Apache Spark for task distribution, significantly reducing the processing time compared with single-machine execution. This paper covers various stages of data collection, preprocessing, visualization, and machine learning applied to both stock and user behavior datasets via PySpark. The incorporation of multiple workers during execution reduced the overall processing time, facilitating more experimental iterations and improving model selection. Future work will investigate additional machine learning models within Spark, such as deep learning, to increase the prediction accuracy and gain more insights in both financial and user behavior datasets.

ABBREVIATIONS

None.

ACKNOWLEDGMENTS

None.

AUTHOR'S CONTRIBUTIONS

All equally contributed to this work, authors read and approved the final manuscript.

FUNDING

None.

AVAILABILITY OF DATA AND MATERIALS

Not applicable.

ETHICS APPROVAL AND CONSENT TO PARTICIPATE

Not applicable.

CONSENT FOR PUBLICATION

Not applicable.

COMPETING INTERESTS

The authors declare that they have no competing interests.

REFERENCES

1. Azeroual O, Fabre R. Processing Big Data with Apache Hadoop in the Current Challenging Era of COVID-19. Big Data Cogn Comput. 2021;5(1):12;Available from: https://doi.org/10.3390/bdcc5010012.
2. Dean J, Ghemawat S. MapReduce: Simplified data processing on large clusters. Commun ACM. 2008;51(1):107-113;Available from: https://doi.org/10.1145/1327452.1327492.
3. Apache Hadoop;Available from: https://hadoop.apache.org/.
4. Zaharia M, Chowdhury M, Das T, et al. Apache Spark: a unified engine for big data processing. Commun ACM. 2016;59(11):56-65;Available from: https://doi.org/10.1145/2934664.
5. Spark* Tuning Guide on 3rd Generation Intel® Xeon® Scalable Processor;Available from: https://www.intel.cn/content/www/cn/zh/developer/articles/guide/spark-tuning-guide-on-xeon-based-systems.html.
6. PySpark Overview - PySpark 3.5.2 documentation;Available from: https://spark.apache.org/docs/latest/api/python/index.html.
7. Preda S, Oprea SV, Bâra A, Belciu A. PV Forecasting Using Support Vector Machine Learning in a Big Data Analytics Context. Symmetry. 2018;10(12):748;Available from: https://doi.org/10.3390/sym10120748.
8. Xianya J, Mo H, Haifeng L. Stock Classification Prediction Based on Spark. Procedia Comput Sci. 2019;162:243-250;Available from: https://doi.org/10.1016/j.procs.2019.11.281.
9. Sarma SLVVD, Sekhar DV, Murali G. Stock market analysis with the usage of machine learning and deep learning algorithms. Bull Electr Eng Informatics. 2023;12(1):552-560;Available from: https://doi.org/10.11591/eei.v12i1.4305.
10. Kalra S, Gupta S, Prasad JS. Predicting Trends of Stock Market Using SVM: A Big Data Analytics Approach. In: Communications in Computer and Information Science, vol. 1229 CCIS. Singapore: Springer; 2020. p. 38-48;Available from: https://doi.org/10.1007/978-981-15-5827-6_4.
11. Nga HTT, Thuy ANT. Recommender System with Apache Spark. In: Lecture Notes in Networks and Systems. Singapore: Springer; 2024. p. 487-497;Available from: https://doi.org/10.1007/978-981-99-6547-2_37.
12. Fernández-Gómez AM, Gutiérrez-Avilés D, Troncoso A, Martínez-Álvarez F. A new Apache Spark-based framework for big data streaming forecasting in IoT networks. J Supercomput. 2023;79(10):11078-11100;PMID: 36845222. Available from: https://doi.org/10.1007/s11227-023-05100-x.
13. Kanchanamala P, Karnati R, Bhaskar Reddy PV. Hybrid optimization enabled deep learning and spark architecture using big data analytics for stock market forecasting. Concurr Comput. 2023;35(8);Available from: https://doi.org/10.1002/cpe.7618.
14. Hope TMH. Linear regression. In: Machine Learning: Methods and Applications to Brain Disorders. Academic Press; 2020. p. 67-81;PMID: 32212548. Available from: https://doi.org/10.1016/B978-0-12-815739-8.00004-3.