European Journal of AI, Computing & Informatics

Vol. 1 No. 2 2025

Article Open Access

Research on Data Analysis Application Based on Spark Computing

Jialu Yan [1,*]

[1] Decoded Advertising, New York, 10005, USA
* Correspondence: Jialu Yan, Decoded Advertising, New York, 10005, USA

Abstract: With the development of the big data era, how to efficiently process and analyze massive data has become an urgent problem that needs to be solved in various industries. Spark, as an open-source distributed computing framework, quickly emerged as a mainstream tool in data processing due to its excellent in memory data processing capabilities and user-friendly interface. Not only does Spark have efficient computing performance, a rich ecosystem, and comprehensive support for data analysis tasks, but it is also widely used in multiple fields such as data management, machine learning, and real-time analysis. This article will delve into the application of Spark in data analysis platforms, reveal the current challenges faced, and explore corresponding optimization approaches, with the aim of significantly improving the efficiency and overall performance of data analysis.

Keywords: Spark computing; big data analysis; data optimization; real time processing; distributed computing

Received: 14 May 2025
Revised: 19 May 2025
Accepted: 11 June 2025
Published: 12 June 2025

1. Introduction

In the context of the big data era, various industries are facing the challenge of how to efficiently process and analyze massive amounts of information. Spark, as an open-source big data distributed computing framework, has become a popular tool in the field of data analysis in a short period of time due to its speed, versatility, and fault tolerance. This article focuses on the application and optimization methods of Spark in big data processing, and analyzes the role of its powerful distributed computing architecture in improving data processing efficiency. The content also covers Spark's system architecture, key modules, as well as its current application status and challenges in practical data analysis. By studying the optimization of data storage and access, memory management strategies, and task scheduling mechanisms, practical ways to improve data analysis efficiency have been proposed. Meanwhile, by citing practical cases such as log analysis, user behavior recommendation, and real-time data stream processing, the practicality and application prospects of Spark in data analysis are further elaborated.

2. Overview of Spark Computing Platform Technology

Spark, as an open-source distributed computing platform for big data processing, plays a crucial role in big data processing due to its excellent memory processing capabilities and efficient task scheduling mechanism. This framework is characterized by modular design and integrates multiple key modules such as Spark Core, Spark SQL, Spark

Streaming, MLlib, and GraphX, which can meet various complex application requirements including batch processing, real-time data stream processing, machine learning analysis, and graph computing. Spark Core is responsible for core distributed task scheduling and memory management tasks, utilizing scalable distributed datasets (RDDs) to improve data processing efficiency and ensure system reliability. The Spark SQL module focuses on querying structured data, Spark Streaming focuses on processing real-time data streams, the MLlib module integrates a distributed machine learning algorithm library, and the GraphX module focuses on processing graph data. Spark's modular structure gives it excellent adaptability and scalability when handling complex data analysis tasks [1,2].

3. Current Status of Data Analysis Applications in Spark

3.1. Performance Issues and Optimization Requirements in Data Processing

When dealing with massive amounts of data, Spark's efficiency is often limited by numerous elements. The huge data scale and complex task logic may cause excessive use of memory, slow processing speed, and in severe cases, memory overflow. Uneven data distribution can result in varying levels of burden on each computing node, thereby affecting overall job efficiency. In addition, frequent Shuffle processes can result in a large amount of data being transmitted between nodes, which undoubtedly slows down the execution speed of tasks. Improper task allocation and resource control may also lead to an increase in task waiting time and a decrease in resource utilization efficiency, which together limit Spark's performance level [2].

3.2. Data Storage and Access Efficiency Issues

In large-scale data processing scenarios, Spark's performance is directly constrained by data storage efficiency, which is one of the decisive factors determining its performance. Diversified data storage formats, such as JSON, Parquet, CSV, etc., result in varying data loading speeds, which undoubtedly have a negative impact on data processing speed. Meanwhile, when Spark adopts HDFS or other distributed storage solutions, the data partitioning strategy and storage structure play a crucial role in the efficiency of task execution and load distribution among nodes [3]. If data partitioning is not reasonable enough, data access lacks index support, I/O operations are too frequent, or unstructured data is not properly optimized, these factors will significantly slow down the speed of data reading and computation, thereby affecting the efficiency of the entire data analysis process and the reliability of results.

3.3. Limitations of Spark in Complex Data Analysis Tasks

Despite Spark's strong computing capabilities in the face of heavy data analysis challenges, its inherent shortcomings cannot be ignored. When dealing with high-dimensional and multi-dimensional data, Spark's memory usage and computational difficulty sharply increase, often leading to performance difficulties. Especially when dealing with complex graph theory calculations, deep learning models, and highly interactive data analysis, Spark seems to lack specific optimization strategies, resulting in low computational efficiency. Moreover, although Spark has shown strong adaptability in batch and stream processing, it may encounter difficulties in task scheduling and resource allocation in complex tasks that require frequent data conversion [4]. For tasks involving highly complex algorithms, Spark's built-in algorithm library and optimization methods may not be fully supported, which undoubtedly increases the difficulty of development and deployment. These limitations pose a severe challenge to performance and processing capabilities when faced with large-scale and complex data analysis tasks [5].

3.4. Data Security and Privacy Issues

Ensuring the security and privacy of data has become a crucial issue in the process of using Spark technology for data processing and analysis. With the increasing amount of information, the urgency of protecting sensitive information is growing. When Spark performs distributed computing tasks, data is transmitted and stored between nodes, and the potential for data leakage during this process cannot be underestimated. Without effective encryption methods and comprehensive access permission management, the confidentiality of data in the flow and storage process between nodes will be difficult to ensure. Meanwhile, Spark's inadequate functionality in data monitoring and abnormal behavior analysis limits the traceability of data operations, thereby increasing the likelihood of data being abused or illegally accessed. For data analysis applications involving a large amount of critical information, it is urgent to strengthen the security and privacy protection functions of the Spark platform [2].

4. Optimization Strategies of Spark in Data Analysis

4.1. Optimization of Data Storage and Access

When using Spark for big data analysis, the efficiency of data storage and access plays a decisive role in the overall system performance. Choosing the appropriate data storage format is crucial, such as column-based storage formats represented by Parquet and ORC, which have efficient column processing capabilities and excellent compression algorithms that can significantly improve data reading efficiency and reduce storage requirements. Traditional line based storage methods such as CSV and JSON often result in slow reads and increased memory usage when dealing with large-scale datasets. Therefore, adopting more efficient data storage formats is an effective means to improve data access efficiency. Meanwhile, a scientific data partitioning scheme is also essential for preventing data skew and balancing node workloads. By implementing partitioning operations on data and ensuring balanced distribution among computing nodes, the parallelism of task processing can be effectively enhanced, and the waiting time of cluster nodes can be shortened [6].

Establishing indexes while storing data can quickly lock in target information and significantly shorten the search process. By utilizing the caching and persistence methods of the Spark platform, such as using the cache() and persist() functions, frequently accessed data can be retained in memory to prevent duplicate reads from disk. This not only speeds up task processing but also significantly reduces input and output expenses. Table 1 provides a detailed display of the differences in reading speed and storage space among different data formats.

Table 1. Comparison of Reading Performance and Storage Usage for Different Data Formats.

Data format	Reading time (seconds)	Compression ratio	Storage space occupation
CSV	one hundred and twenty	nothing	5GB
JSON	one hundred and ten	nothing	4.8GB
Parquet	thirty	high	1.5GB
ORC	thirty-five	high	1.6GB

4.2. Memory Management and Cache

Spark utilizes memory management and caching mechanisms as its core optimization techniques in large-scale data processing. Spark relies on its in-memory computing architecture to achieve fast data access, avoiding frequent disk I/O operations and significantly improving computational efficiency. However, improper memory management may lead to issues such as memory leaks, data corruption, or limited system performance. Proper memory resource allocation becomes crucial, especially when dealing with large datasets. In Spark, memory is mainly divided into two large blocks: storage and execution.

Storage memory is mainly used for caching data, while execution memory is used to retain temporary data during the calculation process (such as data in shuffle operations). Spark can flexibly adjust the allocation of memory usage based on the workload of tasks to achieve efficient use of memory. To prevent memory overflow, setting the total amount of memory and adjusting the proportion of memory allocation reasonably is the core of improving system performance [7].

Spark has rich data caching technologies, including functions such as cache() and persist(). Among them, the cache() function is responsible for temporarily storing data in memory, which is particularly suitable for small-scale datasets with high access frequency. In contrast, the persist() function provides more flexible storage options, allowing users to choose memory, hard drive, or a combination of both to store data according to their needs, making it ideal for handling large-scale datasets or data that needs to be saved for a long time. By cleverly utilizing these caching and persistence techniques, Spark can significantly reduce unnecessary computational costs and greatly improve the execution speed of jobs.

In terms of memory optimization, Spark uses the LRU (Least Recently Used) algorithm to evict less accessed data to ensure maximum utilization of memory resources. For example, in a certain job, the memory requirement for each data partition is M bytes. When the size of the entire dataset exceeds the current memory capacity, the calculation formula for memory allocation can be expressed as:

$$\text{Total memory usage} = \sum_{i=1}^{n} M_i$$

In the case of excessive use of memory resources, Spark will adopt the Least Recently Used (LRU) algorithm to clean up infrequently accessed data in the cache, thereby freeing up necessary memory resources to meet the current computing pressure. By adjusting the memory allocation strategy and efficiently utilizing caching mechanisms, Spark has significantly improved the speed of large-scale data processing, reduced frequent disk read and write operations, and ensured efficient and smooth data processing.

4.3. Task Scheduling and Parallelization Processing

Task scheduling and parallelization play a crucial role in the performance improvement of Spark. Spark splits computing tasks into numerous small execution units (tasks) and uses a DAG scheduler to control the order of task execution, ensuring that tasks are effectively arranged based on their dependencies, and implementing parallel processing in the computing cluster. This processing mode significantly improves computation speed and shortens the time required to complete tasks. Spark implements a strategy called delayed computation, which means that the computation task is not immediately executed at the beginning of creation, but rather a complete execution strategy graph (DAG) is constructed, which only triggers the computation process and carries out the task when certain action operations (such as collect(), save(), etc.) are performed. This method enables Spark to integrate multiple operations together, reducing the storage and transmission requirements of intermediate data, thereby further optimizing execution efficiency.

In parallel processing, the key to efficiency lies in the rationality of data partitioning. Spark technology discretizes data into numerous blocks, which are then distributed to various nodes in the cluster for parallel computation. By appropriately adjusting the number of data blocks and optimizing task allocation, it is possible to effectively prevent data skew, ensure workload balance among nodes, and achieve optimal resource utilization. Appropriate task allocation strategies and parallel processing methods enable Spark to efficiently process large-scale datasets, greatly improving the overall performance of computation.

5. Application of Spark in Actual Data Analysis

5.1. Log Data Analysis and Optimization

In the practical application process of data processing, the analysis of log data constitutes one of the key application scenarios. Many companies rely on collecting and analyzing log data to understand the operating status of their systems, gain insights into user habits, and improve product performance. Spark, as an efficient distributed computing framework, has demonstrated its unique advantages in processing log data. Taking the user log analysis of a well-known e-commerce website as an example, the website generates a massive amount of user behavior records every day, covering a series of behaviors such as browsing, clicking, and purchasing. With the help of Spark technology, the platform can collect and store log information distributed on different servers into HDFS (Distributed File Storage System), and then use Spark SQL and DataFrame interfaces to clean, format, and deeply mine the data. Specifically, Spark performs well in executing SQL queries, efficiently completing tasks such as user access frequency statistics, purchasing behavior research, and click through rate calculations.

Spark's in-memory processing capabilities significantly improve the speed of data processing, especially when dealing with large datasets, effectively bypassing the limitations of disk I/O. Spark utilizes its caching mechanism and partitioning techniques to further improve the efficiency of data reading and processing, ensuring the smooth operation of computing jobs. Through this approach, the platform has achieved real-time analysis of user behavior, and thus optimized product recommendation algorithms and advertising targeting strategies with precision. Through in-depth analysis of logs, e-commerce websites can accurately grasp user needs, enhance user interaction experience and market competitiveness of products. In the context of the big data era, it also ensures the ability to quickly process and analyze real-time information flow.

5.2. User Behavior Analysis and Recommendation System

In the field of contemporary data analysis, Spark technology plays a key role in user behavior analysis and the construction of personalized recommendation systems. With the advancement of big data technology and artificial intelligence, many companies improve product performance and service quality by deeply analyzing user behavior patterns, thereby enhancing user interaction experience. Thanks to Spark's outstanding distributed computing capabilities, it occupies an indispensable position in the analysis of user behavior data and the design of recommendation systems. Taking a streaming music service platform as an example, the platform collects a massive amount of user behavior information every day, such as users' listening records, favorite lists, search habits, etc. In order to create a music recommendation system that meets users' personal preferences, the platform must process this data quickly and accurately. The application of Spark enables the platform to collect and store user data from different servers into HDFS, and use Spark MLlib (machine learning library) to preprocess and analyze this data in depth. In practical operation, Spark can execute recommendation algorithms based on collaborative filtering, identify the similarity between users, and then recommend personalized music content that matches users' preferences.

By leveraging Spark's distributed computing capabilities, it can significantly improve the efficiency of processing large datasets, especially in the stages of user similarity calculation and model training for recommendation systems. The platform can update recommendation algorithms in real-time by deeply analyzing user behavior information, and customize personalized music recommendations for users, including songs, collections, and playlists, in order to improve user engagement and platform user stickiness. Through Spark's performance optimization, the recommendation system not only achieves faster response times but also maintains high efficiency when processing large amounts of data. This results in more accurate recommendations, which enhance user experience and improve the platform's competitiveness in the market.

5.3. Real Time Data Stream Analysis

The real-time data stream analysis function is a core technology of Spark in big data applications, especially in applications that require rapid response and immediate decision-making, demonstrating its unique advantages. Taking the financial sector as an example, real-time tracking of trading information and real-time grasp of market dynamics play a decisive role in avoiding risks and seizing investment opportunities. A well-known securities firm has adopted Spark Streaming technology to implement real-time data stream analysis work, in order to grasp market dynamics and issue risk warnings in a timely manner.

Enterprises utilize the combination of Spark Streaming and Kafka to efficiently capture and process real-time stock market trading information. Every second, the massive amount of data surging out of the stock market includes key elements such as stock prices, trading volumes, and time stamps. These real-time data then flow into the Kafka cluster and are taken over by the Spark Streaming system for real-time data mining. Spark Streaming breaks down continuous streaming data into micro-batches, allowing for in-depth analysis of trading trends, price changes, and precise estimation of the real-time volatility of individual stocks.

In specific business scenarios, Spark Streaming demonstrates powerful real-time big data stream processing capabilities, supporting windowed operations on data streams and gaining insights into stock price trends. Once the fluctuation of a stock price exceeds the preset risk limit, the system will quickly trigger an alarm, prompting traders to take necessary actions. Spark's efficient data processing performance, with its high throughput and minimal latency, ensures that data can be processed within seconds. This enables securities firms to respond rapidly to the market, develop trading strategies, and manage risks in a timely manner. By utilizing Spark Streaming technology, the enterprise has successfully achieved rapid processing and response to real-time transaction data, greatly improving the accuracy of market prediction and risk warning.

6. Conclusion

Research on data processing and analysis using Spark technology demonstrates its strong capabilities and broad application prospects in big data processing. Spark, with its outstanding memory processing capabilities, strong distributed computing performance, and modular system architecture, greatly improves the flexibility and efficiency of data analysis work. In numerous practical application cases, Spark has demonstrated excellent performance in log analysis, user behavior recommendation system, and real-time data stream processing. However, Spark still faces many challenges in terms of data confidentiality, memory resource management, task allocation, and storage efficiency, which urgently require further technological innovation and optimization. With the continuous advancement of technology and the expansion of application fields, Spark will continue to play a crucial role in big data analysis. It will promote the utilization and innovative development of data resources for enterprises and society.

References

1. L. Theodorakopoulos, A. Karras, and G. A. Krimpas, "Optimizing apache spark MLlib: Predictive performance of large-scale models for big data analytics," *Algorithms*, vol. 18, no. 2, p. 74, 2025, doi: 10.3390/a18020074.
2. S. Muvva, "Optimizing Spark data pipelines: A comprehensive study of techniques for enhancing performance and efficiency in big data processing," *J. Artif. Intell. Mach. Learn. Data Sci.*, vol. 1, no. 4, pp. 1862–1865, 2023, doi: 10.51219/JAIMLD/sainath-muvva/412.
3. F. Song, K. Zaouk, C. Lyu, A. Sinha, Q. Fan, et al., "Spark-based cloud data analytics using multi-objective optimization," in *Proc. IEEE 37th Int. Conf. Data Eng. (ICDE)*, 2021, pp. 00041, doi: 10.1109/ICDE51399.2021.00041.
4. M. P. Ramkumar, P. V. B. Reddy, J. T. Thirukrishna, et al., "Intrusion detection in big data using hybrid feature fusion and optimization enabled deep learning based on spark architecture," *Comput. Secur.*, vol. 116, p. 102668, 2022, doi: 10.1016/j.cose.2022.102668.

5. S. Ibtisum, E. Bazgir, S. M. A. Rahman, et al., "A comparative analysis of big data processing paradigms: Mapreduce vs. apache spark," *World J. Adv. Res. Rev.*, vol. 20, no. 1, pp. 1089–1098, 2023, doi: 10.30574/wjarr.2023.20.1.2174.
6. P. Sewal and H. Singh, "A critical analysis of apache hadoop and spark for big data processing," in *Proc. 6th Int. Conf. Signal Process., Comput. Control (ISPCC)*, 2021, doi: 10.1109/ISPCC53510.2021.9609518.
7. M. Babar, M. A. Jan, X. He, M. U. Tariq, et al., "An optimized IoT-enabled big data analytics architecture for edge–cloud computing," *IEEE Internet Things J.*, vol. 10, no. 5, pp. 3995–4005, 2022, doi: 10.1109/JIOT.2022.3157552.