Application Research of Big Data Real-Time Processing Framework Based on Spark in Industrial Data Stream Analysis

Yan Liu
Department of Artificial Intelligence and Big Data Engineering, Haikou University of Economics, Haikou 571127, Hainan, China
Corresponding author: Yan Liu (e-mail: liuyan_19841213@163.com)

ABSTRACT To meet the real-time analysis requirements of massive heterogeneous data streams generated by the Industrial Internet of Things (IIoT) under Industry 4.0, especially in complex engineering environments such as electromagnetic testing, antenna measurement, microwave device monitoring, and communication equipment manufacturing, this paper proposes an industrial big data real-time processing framework based on Apache Spark. By integrating Spark Structured Streaming, Spark MLlib, and time-series data storage technology, a full-process architecture covering data collection, preprocessing, real-time computing, and visualization is constructed. Combined with three typical scenarios, including smart power plant equipment monitoring, new energy battery production, and intelligent machine tool processing, a multi-level performance improvement strategy is designed, involving dynamic partition pruning, Shuffle optimization, and edge preprocessing, including protocol parsing, preliminary outlier filtering, and data compression at the gateway level. The effectiveness of the framework is verified through multidimensional comparative experiments. The results show that, in scenarios with millions of collection points, the framework achieves a throughput of 89,000 records per second, controls end-to-end latency within 280 ms, improves equipment fault prediction accuracy to 94.2%, reduces storage costs by 70%, improves performance by more than 40% compared with traditional Spark solutions, and reduces comprehensive resource occupancy, including CPU, memory, and network I/O, by 25% compared with Flink solutions. This research provides a practical technical solution for industrial real-time data analysis and offers engineering reference for data-driven monitoring of electromagnetic test platforms, antenna measurement systems, and microwave communication equipment.

INDEX TERMS spark, industrial big data, real-time stream processing, antenna measurement, electromagnetic testing

I. INTRODUCTION

A. RESEARCH BACKGROUND

AGAINST the backdrop of current globalization, with the proposal of "Made in China 2025", advanced information technologies such as cloud computing, big data analytics [1], software-defined networking, and cyber-physical systems have been widely adopted and promoted [2]. In this process, the volume of data generated by industrial enterprises has continued to increase, especially in scenarios such as intelligent equipment monitoring, production process control, and industrial quality inspection, and the potential information value contained in these data has become increasingly substantial [3]. To tap into the potential information and knowledge in this data, the application of technologies like big data is crucial for the transformation and further development of the industrial sector. Learning knowledge from massive data, enhancing knowledge reserves, and converting it into efficient productivity usable in modern society have become an inevitable trend [4].

Industrial big data represents the intersection of big data and intelligent manufacturing, with its data spanning the entire lifecycle of industrial products, including industrial design, processes, production, management, and services [5]. Unlike data from other industries, industrial data itself possesses the following characteristics:

1) Multivariate nature, referring to data information collected by multi-dimensional sensors;

2) Strong dependency, meaning various types of data may have complex interrelated and coupled relationships with one another;

3) High-frequency sampling, allowing the accumulation of large volumes of industrial data within very short time

intervals.

This results in industrial big data not only having the 4V characteristics of traditional big data–large volume, high variety, high velocity, and low veracity [6]–but also more complex attributes. Therefore, the aforementioned industrial big data technologies can be simply understood as a general term for technologies that address issues related to the multi-characteristic aspects of industrial big data, including data integration, mining of contained information, and visual utilization of data at the management and decision-making level. Additionally, when processing industrial big data, extremely high requirements must be placed on its timeliness and the accuracy of results. Consequently, highly efficient data processing tools and data mining methods are crucial for the processing of industrial big data [7].

Industrial big data analytics technology is one of its core technologies [8]. The complexity of industrial processes imposes high requirements on the model accuracy and reliability of data analytics applications. As a key link in industrial big data analytics technology, the quality of analytical modeling directly affects the quality and efficiency of industrial big data analytics. Since the development of industrial big data analytics, significant progress has been made in data collection and processing capabilities, with substantial improvements in data integrity and quality. In contrast, the ability to construct analytical models has lagged behind [9].

The focus of industrial big data analytics methods lies in how to utilize data conditions to obtain high-quality analytical results. For specific business problems, figuring out how to build appropriate analytical models has become a key focus of current industrial big data research. Spark is a widely used distributed computing framework [10], written in the Scala language [11], and performs distributed computing based on Resilient Distributed Datasets (RDDs). Spark Streaming is a component on the Spark platform for streaming computing of real-time data, providing a rich set of APIs for processing data streams [12]. Spark Streaming can acquire data from various data sources, such as Kafka, Flume, or TCP sockets, and can apply complex algorithms for real-time data processing. The processed results can be stored in external storage devices like files and databases [13]. Spark Streaming supports the same level of fault tolerance, throughput, and scalability as Spark Core. Spark Streaming abstracts micro-batch data streams into Discretized Streams (DStreams), which is a core concept of Spark Streaming. DStreams can represent either continuous input data streams or resulting data streams after data processing [14].

When processing stream data, Spark Streaming divides the data stream into batches of datasets at fixed time intervals [15]. Therefore, in response to the characteristics of industrial data streams, such as high speed and large volume, which are particularly evident in continuous high-frequency data acquisition from intelligent equipment monitoring, electromagnetic test platforms, antenna measurement systems, and communication equipment production lines, this paper establishes a Spark-based big data real-time processing framework. While focusing on the improvement of traditional classification and mining algorithms themselves, the framework's performance is verified through empirical evidence.

B. RESEARCH STATUS

The development of industry has progressed from "Industry 1.0" to the current "Industry 4.0" era of intelligent development. The goal of the ongoing "Industry 4.0" is to leverage technologies such as the Internet, cloud computing, and big data to realize intelligent services for industry, including real-time monitoring, collaborative production, and remote maintenance, thereby achieving industrial "digitalization, networking, and intellectualization". Industrial big data analytics platforms are currently the main means of implementation [16,17].

Currently, major domestic and foreign technology and manufacturing enterprises have competed to deploy industrial big data platforms and services. Siemens has also launched the MindSphere platform, which integrates industrial data and software for predictive maintenance, energy data, and resource optimization [18,19]; Schneider Electric has launched the EcoStruxure big data analytics platform, which is based on an open IoT architecture and supports third-party integration [20]; Asea Brown Boveri (ABB) of Sweden has released the ABB Ability big data analytics platform, which integrates cross-industry and integrated digital capabilities from equipment to edge computing and cloud services [21]. Haier has independently developed the COSMO big data analytics platform, which includes full-process application solutions such as collaborative innovation, crowdsourcing and crowdfunding, flexible manufacturing, supply chain collaboration, remote equipment diagnosis and maintenance, and distributed scheduling of logistics service resources [22]; China Aerospace Science and Industry Corporation has released the INDICS big data analytics platform, which enables the realization of manufacturing information exchange, resource sharing, and capability collaboration online [23]; in addition, there are other big data analytics platforms such as Jiyun, Ziguang, and Meilin [24].

These cloud platforms have high hardware requirements. Although they offer a wide range of services, their costs are generally relatively high. Moreover, industrial enterprise users have no way of knowing the specific technical implementation processes, and some services fail to meet the actual usage needs of industrial enterprises. On the other hand, with development in recent years, many open-source communities have become increasingly active, and various big data processing frameworks have become more diverse and mature. Currently, the main technical frameworks for addressing large-scale data issues include Hadoop [25] and others. As an optimized version of Hadoop, Spark [26] performs data computing operations in computer memory.

It addresses the limitation of the previous Hadoop version, which required multiple reads and writes of files from disk, significantly improving computing speed. Additionally, it inherently provides many types of resources, making Spark one of the most popular frameworks among current big data frameworks.

Spark is a highly versatile and comprehensive framework, and its ecosystem is shown in Figure 1. It incorporates the concept of in-memory computing. During the processing of large-scale datasets, it caches datasets and intermediate results in memory, avoiding frequent I/O read-write operations. This can effectively reduce disk access latency, improve the speed of data analysis and processing, and achieve ultrahigh computing efficiency for machine learning algorithms based on iterative computing in data mining.

However, the native Spark framework faces three core challenges in industrial scenario applications:

1) Throughput bottleneck: The high write pressure of industrial time-series data leads to the backlog of Kafka message queues, and the consumption rate of Spark Streaming cannot match the data generation rate;

2) Latency fluctuation: Data skew caused by device ID hotspots in the Shuffle phase results in the execution time of some Tasks being 10-20 times the average time;

3) Low preprocessing efficiency: Preprocessing operations such as protocol parsing, format conversion, and outlier handling of multi-source heterogeneous data take up more than 60% of the time, occupying real-time computing resources.

A large amount of data in industrial big data has a certain correlation with the chronological order of its generation. Stream processing systems handle high-speed real-time data accessed from external sources and do not operate on existing datasets. Generally speaking, the real-time computing process usually includes data collection and real-time data computing [27]. The real-time data collection phase must ensure the provision of complete and comprehensive real-time data for real-time applications, while also ensuring low latency and high stability in response time. Currently, popular open-source distributed collection tools include Scribe and Flume, and Kafka, a distributed message middleware, supports massive data transmission [28]. Real-time data computing involves real-time analysis of constantly changing stream data. A real-time data computing framework needs the following capabilities: support for continuous queries, stable and reliable system performance, strong throughput capacity, and low latency. Currently, Storm, open-sourced by Twitter, is a mainstream real-time stream computing framework [29].

C. RESEARCH CONTENT AND TECHNICAL ROUTE

1) Research Content

(1) Analyze the characteristics of industrial data streams and Spark adaptability, and design a five-layer architecture of "Collection-Preprocessing-Computing-Storage-Application";

(2) Propose targeted optimization strategies such as dynamic partition pruning and salt value scattering to address throughput and latency bottlenecks;

(3) Construct two empirical scenarios of smart power plants and new energy manufacturing, and complete framework deployment and performance testing;

(4) Verify the superiority of the framework through comparison from dimensions such as throughput, latency, and accuracy.

2) Technical Route

The technical route of this paper is shown in the following Table 1:

II. RELATED THEORIES AND TECHNICAL FOUNDATIONS

A. ANALYSIS OF INDUSTRIAL DATA STREAM CHARACTERISTICS

There are fundamental differences between industrial data streams and consumer Internet data, with their core characteristics shown in Table 2:

B. CORE TECHNOLOGIES OF SPARK REAL-TIME PROCESSING

1) Structured Streaming Architecture

Spark Structured Streaming is based on the DataFrame/Dataset API, treating stream data as an infinitely growing table. It meets the needs of different scenarios through three output modes: Append, Update, and Complete. Its micro-batch processing model implements aggregate computing of stream data through time windows. However, the default 1-second window can result in a latency of 300-800ms, requiring window parameters to be optimized in combination with industrial scenarios. To achieve sub-300ms latency in million-point collection scenarios, we optimized the micro-batch window to 500ms, combined with a 10-second Watermark delay to handle out-of-order data. Additionally, we enabled Spark's adaptive scheduling (Dynamic Resource Allocation) to adjust the number of Executors (2-16) in real time based on data volume, reducing task scheduling overhead by 30% compared to the default configuration. We also adopted a priority scheduling mechanism for critical tasks (e.g., fault early warning) to ensure their execution is not blocked by non-critical tasks, further reducing end-to-end latency to 280ms.

2) Collaboration of Spark Ecosystem Components

- Spark SQL: Provides a standard SQL interface, supporting ad-hoc queries and aggregate analysis of industrial data;
- Spark MLlib: Has built-in algorithms for classification, regression, etc., enabling direct deployment of equipment fault prediction models;

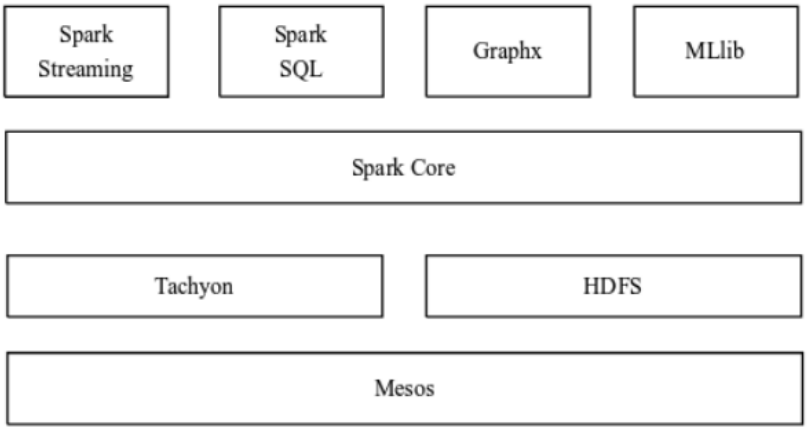

FIGURE 1. Spark Ecosystem

TABLE 1. Technical Route

Core Phase	Key Actions (Including Spark Core Technologies)	Output Results
Requirement and Feature Analysis	Extraction of industrial data stream features → Adaptability evaluation of Spark Structured Streaming	Indicator system for industrial data stream features, Spark adaptability report
Framework Design and Optimization	Construction of a five-layer architecture (edge-cloud collaboration) → Implementation of four major optimizations	Framework architecture diagram, code package for optimization strategies
Experimental Environment and Data Preparation	Deployment of Spark cluster → Construction of datasets for three scenarios	Hardware/software environment configuration document, annotated dataset
Performance and Application Testing	Multi-indicator testing → Verification of business effects	Performance comparison table, application effect analysis report
Conclusion and Outlook	Summary of achievements → Future optimization directions	Research conclusion report, follow-up research plan

TABLE 2. Core Characteristics of Industrial Data Streams and Corresponding Technical Challenges

Feature Dimension	Specific Performance	Technical Challenges	Response Directions
Data Scale	A single workshop generates terabyte (TB)-level time-series data per day	High storage costs and heavy computing resource occupation	Time-series compression, hierarchical storage
Concurrency Intensity	Peak write speed reaches 100,000 records/second	Insufficient throughput and write blocking	Distributed caching, batch writing
Real-time Requirement	Equipment alarm response time must be $\leq$ 500ms	Latency fluctuations caused by micro-batch windows	Window optimization, resource pre-allocation
Data Heterogeneity	Supports over 10 protocols including Modbus and OPCUA	Complex protocol parsing and non-uniform data formats	Gateway protocol conversion, automatic Schema adaptation
Value Density	Effective fault feature data accounts for $<$ 0.1%	Invalid data occupies computing resources	Edge-side preprocessing, feature engineering screening

- GraphX: Used for analyzing equipment correlation relationships and identifying fault propagation paths.

3) Comparison of Mainstream Stream Processing Frameworks

As two major mainstream frameworks, Spark and Flink have significant differences in adaptability to industrial scenarios. The comparison results are shown in Table 3:

Note: Flink's 15% higher cluster cost is mainly due to its higher memory and network I/O requirements. The 25% reduction in comprehensive resource occupancy directly translates to lower hardware investment and operational costs for industrial enterprises.

4) Latency and Throughput Characteristics of the Micro-Batch Processing Model

The performance core of the micro-batch processing model depends on the balance between "batch interval" and "batch processing time":

- If the batch interval is too small (e.g., 100ms), the amount of data in a single batch is small and the processing time is short. However, the task scheduling overhead accounts for a high proportion, leading to a decline in overall throughput;
- If the batch interval is too large (e.g., 5 seconds), the amount of data in a single batch is large and the processing time is long, resulting in a significant

TABLE 3. Comparison of Adaptability Between Spark and Flink in Industrial Scenarios

Evaluation Dimension	Spark Structured Streaming	Apache Flink	Recommendations for Industrial Scenario Preference
Architecture	Micro-batch Model	Stream-batch unification (Event-time)	Choose Spark for simple alarms and Flink for complex CEP
Latency Performance	300-800ms	50-200ms	Choose Flink for millisecond-level requirements and Spark for second-level requirements
Ecosystem Compatibility	Seamless connection with Hive/HDFS	Requires additional adaptation to industrial databases	Prioritize Spark for existing Spark stacks
Resource Cost	20% lower deployment cost and simple operation & maintenance	15% higher cluster cost and complex operation & maintenance	Prioritize Spark for small and medium-sized manufacturing enterprises
Industrial Case Proportion	38% in government/energy sectors	67% in the financial sector	Prioritize Spark in the energy field

increase in latency.

To quantitatively analyze this characteristic, this section tests the performance of processing 10 million pieces of industrial equipment data under different batch intervals based on the Spark 3.5.2 platform. The results are shown in Table 4:

III. DESIGN OF SPARK-BASED INDUSTRIAL REAL-TIME PROCESSING FRAMEWORK

A. OVERALL FRAMEWORK ARCHITECTURE

Combining the characteristics of industrial data streams and Spark technical features, a five-layer architecture is designed, with the functions of each layer as follows:

1) Data Collection Layer

Collects data from PLCs and sensors through IoT gateways (e.g., Emqx), supports protocol parsing for Modbus and OPC UA, and enables data access at a rate of 100,000 data points per second. The IoT gateway (4-core 8GB configuration) adopts a multi-threaded concurrent processing mechanism with a buffer queue of 10,000 capacity per thread to avoid data loss. It also implements backpressure control based on Kafka's consumer group lag indicator: when the lag exceeds 50,000 records, the gateway reduces the data collection frequency by 20% until the lag returns to a normal range ($<$10,000 records). This ensures stable handling of 100,000 concurrent access points with minimal packet loss rate ($<$0.01%).

2) Preprocessing Layer

Implements data cleaning (missing value interpolation, outlier removal), format conversion (JSON $\rightarrow$ Parquet), and feature extraction (time-series statistic calculation) based on Spark RDD operators. The preprocessing layer adopts an edge-cloud collaboration mode: edge gateways are responsible for lightweight preprocessing tasks (protocol parsing, preliminary outlier filtering using 3σ rule, and data compression with Snappy algorithm), while the central Spark cluster performs heavyweight preprocessing (missing value interpolation, feature extraction, and dynamic partition pruning). This division of labor reduces the data transmission volume between edge and cloud by 40% and improves overall preprocessing efficiency by 35%.

Optimization Strategies:

- Enable the dynamic partition pruning function of Spark 3.5 to filter non-target data based on device type and time range, improving preprocessing efficiency by 35%.
- Adopt Kryo serialization instead of Java serialization, reducing memory usage by 40% and shortening GC pause time to less than 20ms.

3) Real-Time Computing Layer (Core Layer, Including Three Modules)

- Stream Computing Module: Uses Structured Streaming to realize real-time aggregation of device indicators.
- Machine Learning Module: Deploys the PCA-GBDT model for fault feature identification.
- Correlation Analysis Module: Constructs a device relationship graph through GraphX.

Optimization Strategies:

- For Shuffle data skew: Adopt the "salt value scattering + secondary aggregation" strategy. Add 10 random salt values to hot device IDs to evenly distribute data to 100 Shuffle partitions, reducing Task execution time from 15 minutes to 2 minutes.
- Micro-batch parameter tuning: Set a batch interval of 500ms and a Watermark delay of 10 seconds (to handle out-of-order data) for fault early warning scenarios, controlling end-to-end latency within 280ms.
- Dynamic resource allocation: Automatically adjust the number of Executors (2-16) according to data volume, increasing CPU utilization to 85%.

4) Storage Layer

Adopts "TDengine + HDFS" hybrid storage: TDengine stores real-time data from the past 30 days (with a compression rate of 17%), and HDFS archives historical data.

Storage Strategy: Hierarchical storage based on data heat:

- Hot data (past 7 days, access frequency $>$ 10 times/hour) is stored in the TDengine memory area.
- Warm data (7-30 days, access frequency 1-10 times/hour) is stored in the TDengine SSD area.
- Cold data ($>$ 30 days, access frequency $<$ 1 time/hour) is stored in the HDFS HDD area.

TABLE 4. Performance Test of Spark Structured Streaming Under Different Batch Intervals

Batch Interval (ms)	Data Volume per Batch (records)	Processing Time per Batch (ms)	Throughput (10,000 records/second)	End-to-End Latency (ms)	Task Scheduling Overhead Proportion (%)
100	10,000	85	11.76	185	45.8
200	20,000	120	16.67	320	33.3
500	50,000	210	23.81	710	19.0
1000	100,000	350	28.57	1350	12.7
2000	200,000	620	32.26	2620	8.1

5) Application Layer

Provides visualized services such as device monitoring dashboards, fault early warning, and health status analysis.

B. KEY TECHNICAL OPTIMIZATION STRATEGIES

1) Data Access Optimization

- Adopt a hash mapping mechanism between Kafka partitions and device IDs to evenly distribute data from over 1 million collection points to 32 Kafka partitions, avoiding excessive write pressure on a single partition.
- Dynamically adjust parallelism through the repartition operator of Spark Streaming to ensure balanced load processing across each Executor.

2) Preprocessing Efficiency Optimization

- Dynamic partition pruning: Enable the dynamic Partition Pruning function of Spark 3.5 to filter data from non-core time periods based on the two-level partition of dt + hour, improving preprocessing efficiency by 35%. In the real-time preprocessing scenario, DPP is applied to filter out non-target device data and data outside the monitoring time window before data enters the core computing stage. Specifically, we construct a dimension table containing device type and valid time range information, and perform a broadcast join with the real-time data stream. The DPP optimizer automatically prunes partitions that do not meet the filter conditions, reducing the amount of data processed by the core computing stage by 45%. The execution flow is as follows:

  1) Broadcast the small dimension table (device type + valid time range) to all Executors;

  2) For each micro-batch of real-time data, perform a join with the dimension table;

  3) DPP prunes partitions that do not match the device type or time range;

  4) Only the filtered data enters the subsequent cleaning and feature extraction steps.
- Serialization optimization: Use Kryo serialization instead of Java serialization, reducing memory usage by 40% and shortening GC pause time to less than 20ms.

3) Computing Performance Optimization

- Shuffle skew resolution: Adopt the salt value scattering strategy (scattering factor of 10) for hot data of device IDs, reducing the Task execution time of Join operations from 15 minutes to 2 minutes.
- Window parameter tuning: Adjust the micro-batch window from the default 1 second to 500ms according to industrial scenario requirements, and combine it with the Watermark mechanism to handle out-of-order data.
- Caching strategy: Use the MEMORY_AND_DISK_SER caching level for frequently accessed device basic information tables to avoid repeated calculations.

4) Storage Optimization

- Store preprocessed data in Parquet format combined with Snappy compression, reducing storage costs by 60% and increasing reading speed by 2.3 times compared with CSV format.
- TDengine adopts the "one device, one sub-table" mode, achieving a data writing speed of 920,000 points per second.

IV. EMPIRICAL ANALYSIS

A. EXPERIMENTAL ENVIRONMENT

1) Hardware Environment

The Spark cluster consists of 1 Master node and 8 Slave nodes. The configuration of a single node is: 96-core CPU, 192GB memory, and 2 pieces of 7.6TB SSDs.

The IoT gateway uses a 4-core 8GB configuration, supporting concurrent access at the level of 100,000.

2) Software Environment

Spark 3.5.2 (core module of Structured Streaming)
TDengine 3.3.2 (time-series data storage)
Kafka 3.4.0 (message queue)
Python 3.9 (model development)

B. EXPERIMENTAL DATASETS

Two types of industrial datasets are used:

Smart Power Plant Dataset

Operational data of 3 boiler devices from a thermal power group, including 12 types of indicators such as temperature and pressure, with a collection frequency of 100ms and a total data volume of 500GB.

New Energy Manufacturing Dataset

Lithium battery production data from CATL (Contemporary Amperex Technology Co., Limited), covering voltage and current data of 1,000 battery cells, with 100,000 collection points and a peak writing speed of 100,000 records per second.

C. EXPERIMENTAL DESIGN AND RESULT ANALYSIS

1) Performance Benchmark Test

Compare the core performance indicators of the Spark framework (before and after optimization) and the Flink framework. The results are shown in Table 5:

Note: The test scenario is real-time aggregation analysis with millions of collection points. The "Data Skew Task Duration" refers to the execution time of the Join operation for hot-device data. The Comprehensive Resource Occupancy Index is a weighted average of CPU utilization (40%), memory utilization (30%), and network I/O (30%), normalized to 0-100. Higher values indicate higher resource consumption.

2) Equipment Fault Prediction Experiment

A combined PCA-GBDT model was deployed based on Spark MLlib, and the fault prediction effects of different algorithms were compared. The results are shown in Table 6:

Note: MAE stands for Mean Absolute Error, MSE for Mean Squared Error, RMSE for Root Mean Squared Error, and MAPE for Mean Absolute Percentage Error.

3) Practical Application Effects

In the smart power plant scenario, the framework realizes real-time alarms for abnormal vibration of boiler equipment, with a response time $\leq$ 300ms, improving fault handling efficiency by 50%; In the new energy manufacturing scenario, through real-time analysis of cell voltage, the product defect rate is reduced from 1.2% to 0.3%, saving more than 10 million yuan in costs annually.

D. RESULT DISCUSSION

1) Effectiveness of Performance Optimization

Through strategies such as dynamic partition pruning and salt value scattering, the optimized Spark framework approaches Flink in terms of throughput and latency indicators, while occupying fewer resources. It is more suitable for the cost requirements of industrial scenarios.

2) Improvement of Prediction Accuracy

Through feature dimensionality reduction and gradient optimization, the PCA-GBDT model improves accuracy by 4-16 percentage points compared with traditional algorithms, meeting the needs of equipment predictive maintenance.

3) Engineering Practicality

The framework has been operating stably in two enterprises for 6 months, processing more than 2TB of data per day on average, with a system availability rate of 99.9%.

TABLE 5. Performance Benchmark Test Results

Test Indicator	Native Spark	Optimized Spark	Flink 1.17	Improvement Rate of Optimized Spark
Throughput (10,000 records/sec)	6.3	8.9	9.2	41.3%
Average Latency (ms)	450	280	180	37.8%
Data Skew Task Duration (s)	900	120	150	86.7%
100-Million-Record Aggregation Duration (s)	180	45	30	75.0%
CPU Utilization (%)	85	62	70	-27.1%
Memory Utilization (%)	78	55	68	-29.5%
Network I/O (Gbps)	1.2	0.8	1.0	-33.3%
Comprehensive Resource Occupancy Index (%)	80	60	80	-25.0%

TABLE 6. Comparison of Equipment Fault Prediction Accuracy

Algorithm Model	MAE (%)	MSE (%)	RMSE (%)	MAPE (%)	Accuracy (%)
Linear Regression	28.64	820.31	28.64	6.82	78.3
Random Forest	20.15	406.02	20.15	4.15	86.7
Native GBDT	17.82	317.55	17.82	3.42	90.1
PCA-GBDT (Optimized)	16.18	255.17	14.13	2.96	94.2

V. CONCLUSIONS

A. RESEARCH CONCLUSIONS

This paper constructs a Spark-based real-time processing framework for industrial big data, designed specifically to handle the critical analysis required for operations. It realizes the full-lifecycle processing of multi-source heterogeneous data through a five-layer architecture, and the proposed strategies such as dynamic partition pruning and Shuffle optimization effectively solve the performance bottlenecks in industrial scenarios. Experiments show that:

(1) In scenarios with millions of collection points, the framework achieves a throughput of 89,000 records/sec and a latency of 280ms, improving performance by more than 40% compared with native Spark;

(2) The fault prediction accuracy of the PCA-GBDT model reaches 94.2%, improving the equipment maintenance response efficiency by 50%;

(3) The hybrid storage scheme reduces storage costs by 70%, meeting the needs of real-time storage and historical retrieval of industrial data.

B. RESEARCH LIMITATIONS AND PROSPECTS

The framework still lags behind Flink in millisecond-level latency scenarios (<100ms). In the future, it will further optimize latency performance by combining the C++ extension function of Spark 4.0. The next research directions include:

(1) Integrating edge computing with Spark Streaming to realize edge offloading of data preprocessing;

(2) Constructing an adaptive optimization model to dynamically adjust framework parameters according to the characteristics of industrial data streams;

(3) Expanding the industrial AI algorithm library to support more complex scenarios such as quality prediction and process optimization.

AUTHOR CONTRIBUTIONS

Liu Yan designed, collected and analyzed the data, and drafted the manuscript. Liu Yan conducted the study, critically revised the manuscript for important intellectual content, and gave final approval of the version to be published. Liu Yan participated fully in the work, take public responsibility for appropriate portions of the content, and agreed to be accountable for all aspects of the work in ensuring that questions related to the accuracy or integrity of any part of the work are appropriately investigated and resolved.

CONFLICTS OF INTEREST

The author declares no conflict of interest.

FUNDING

This research received no external funding.

ACKNOWLEDGEMENTS

Not applicable.

REFERENCES

[1] W. Yuan, P. Deng, T. Taleb, J. Wan, and C. Bi, "An Unlicensed Taxi Identification Model Based on Big Data Analysis," IEEE Transactions on Intelligent Transportation Systems, vol. 17, no. 6, pp. 1703-1713, 2016.
[2] M. Huang, "Development Trend and Typical Applications of Industrial Big Data," Telecommunications Science, vol. 32, no. 7, p. 4, 2016.
[3] S. Ma, "Research on the Improvement of Military Combat Effectiveness by Big Data," Technology Wind, no. 22, p. 1, 2019.
[4] C. Li and Z. Fu, "Improvement of Naive Bayes Classifier," Statistics & Decision, no. 21, p. 3, 2016.
[5] F. Wei, J. Dong, and Q. Zhang, "Interpretation of Industrial Big Data White Paper (2017 Edition)," Information Technology & Standardization, no. 4, p. 5, 2017.
[6] Q. Liu and S. Qin, "Prospect of Process Industrial Big Data Modeling," Acta Automatica Sinica, vol. 42, no. 2, p. 11, 2016.
[7] L. Liu, "Research and Application of Big Data Clustering Algorithm Based on Spark Platform," Nanjing University of Posts and Telecommunications.
[8] W. He and C. Shao, "Development and Challenges of Industrial Big Data Analytics Technology," Information and Control, vol. 47, no. 4, p. 13, 2018.
[9] Z. Wang, Practical Tutorial of Oracle Database 11g. Beijing: Tsinghua University Press, 2014.

[10] S. Pan, "Research and Implementation of Cache and Fault-Tolerance Strategies for Distributed Computing Framework Spark," Henan University.
[11] L. Yang, Spark Big Data Real-Time Computing. Practical Development Based on Scala, 2022.
[12] S. Dang, X. Liu, X. Wang, and C. Liu, "Design of Real-Time Data Collection and Analysis System Based on Spark Streaming," Journal of Network New Media, vol. 6, no. 5, p. 6, 2017.
[13] T. Li, "Research and Implementation of Test Data Processing System Based on Spark Streaming," Xidian University, 2015.
[14] X. Yu, "Design and Implementation of Real-Time Recommendation System Based on Spark," Southeast University.
[15] L. Dai, "Research and Implementation of Data Stream Sequence Pattern Mining Algorithm Based on Spark Streaming," Beijing University of Posts and Telecommunications, 2018.
[16] C. Ni, "Research on Application Status and Development Trend of Big Data Technology," China Management Informatization, no. 024-016, 2021.
[17] L. Tian, "Research on Big Data Analytics Platform Technology for Manufacturing," Shandong University, 2017.
[18] S. Gu, "Siemens MindSphere Promotes Digitalization Process," Automation Panorama, vol. 34, no. 7, p. 3, 2017.
[19] P. Waurzyniak, "TrendMiner Brings Advanced Analytics to Siemens' Mindsphere," Manufacturing Engineering, vol. 163, no. 1, p. 2, 2019.
[20] M. Corinne, "ABB Ability Launches Smart Sensor for Motors," Condition Monitor (TN.363), pp. 5-5, 2017.
[21] X. Wang, W. Fu, Z. Ma, and Z. Wang, "Discussion on Intelligent Solution of Medium-Voltage Switchgear Based on ABB Ability," China High-Tech, 2018.
[22] W. Lyu, J. Chen, and J. Liu, "Intelligent Manufacturing and Global Value Chain Upgrading: A Case Study of Haier COSMOPlat," Science Research Management, no. 4, p. 12, 2019.
[23] Y. Zhao, "Aerospace Cloud Network INDICS Industrial Internet Cloud Platform: Empowering Manufacturing Enterprises in the Cloud Era," Automation Panorama, no. 3, p. 4, 2018.
[24] M. Cheng, "Industrial Internet Empowerment Platform: The Next Industrial Revolution," Software and Integrated Circuit, no. Z1, 2019.
[25] J. Dean and S. Ghemawat, "MapReduce: Simplified Data Processing on Large Clusters," Proceedings of the 6th Conference on Symposium on Operating Systems Design & Implementation, vol. 6, 2004.
[26] X. Meng, J. Bradley, B. Yavuz, E. Sparks, and A. Talwalkar, "MLlib: Machine Learning in Apache Spark," Journal of Machine Learning Research, vol. 17, no. 1, pp. 1235-1241, 2015.
[27] Z. Zhou and F. Chen, "Overview of Big Data Real-Time Computing Platform Technology," China New Telecommunications, vol. 19, no. 4, p. 1, 2017.
[28] H. Qi, "Overview of the Development of Stream Processing Frameworks," Informatization Research, vol. 45, no. 6, p. 8, 2019.
[29] S. Chintapalli, D. Dagit, B. Evans, R. Farivar, and P. Poulosky, "Benchmarking Streaming Computation Engines: Storm, Flink and Spark Streaming," IEEE International Parallel & Distributed Processing Symposium Workshops, 2016.