applied sciences

Article

The Proposal and Validation of a Distributed Real-Time Data Management Framework Based on Edge Computing with OPC Unified Architecture and Kafka

Daixing Lu [1,2], Kun Wang [1], Yubo Wang [3] and Ye Shen [2,*]

[1] School of Mechanical Engineering, Shanghai Institute of Technology, Shanghai 201418, China
[2] Haina-Intelligent Manufacturing Industrial Software Research Center, Yangtze Delta Region Institute of Tsinghua University, Jiaxing 314006, China
[3] Department of Mechanical Engineering, RheinMain University of Applied Sciences, 65428 Rüsselsheim, Germany
* Correspondence: shenye@tsinghua-zj.edu.cn

Abstract: With the advent of Industry 4.0, the manufacturing industry is facing unprecedented data challenges. Sensors, PLCs, and various types of automation equipment in smart factories continue to generate massive amounts of heterogeneous data, but existing systems generally have bottlenecks in data collection standardization, real-time processing capabilities, and system scalability, which make it difficult to meet the needs of efficient collaboration and dynamic decision making. This study proposes a multi-level industrial data processing framework based on edge computing that aims to improve the response speed and processing ability of manufacturing sites to data and to realize real-time decision making and lean management of intelligent manufacturing. At the edge layer, the OPC UA (OPC Unified Architecture) protocol is used to realize the standardized collection of heterogeneous equipment data, and a lightweight edge-computing algorithm is designed to complete the analysis and processing of data so as to realize a visualization of the manufacturing process and the inventory in a production workshop. In the storage layer, Apache Kafka is used to implement efficient data stream processing and improve the throughput and scalability of the system. The test results show that compared with the traditional workshop, the framework has excellent performance in improving the system throughput capacity and real-time response speed, can effectively support production process judgment and status analysis on the edge side, and can realize the real-time monitoring and management of the entire manufacturing workshop. This research provides a practical solution for the industrial data management system, not only helping enterprises improve the transparency level of manufacturing sites and the efficiency of resource scheduling but also providing a practical basis for further research on industrial data processing under the "edge-cloud collaboration" architecture in the academic community.

Keywords: intelligent manufacturing; OPC UA; edge computing; high concurrency; real time

check for updates

Academic Editor: Atsushi Mase

Received: 24 April 2025
Revised: 16 May 2025
Accepted: 20 May 2025
Published: 18 June 2025

Citation: Lu, D.; Wang, K.; Wang, Y.; Shen, Y. The Proposal and Validation of a Distributed Real-Time Data Management Framework Based on Edge Computing with OPC Unified Architecture and Kafka. *Appl. Sci.* **2025**, *15*, 6862. https://doi.org/10.3390/app15126862

1. Introduction

Industry 4.0, first proposed in Germany, aims to create new intelligent factories; integrate traditional factories with intelligent equipment, the Internet of Things, artificial intelligence, big data technology, and cloud computing; and promote the transformation of the traditional manufacturing industry toward flexible manufacturing, green manufacturing, and intelligent manufacturing [1].

With the support of digital twins, CPS, intelligent technology, big data technology, and cloud computing, virtual modeling of the production line in a manufacturing workshop can be achieved, and the production line can be monitored in real time using big data technology, truly realizing the industrial potential of the manufacturing factory [2]. Through research on intelligent technology for factories, the manufacturing efficiency of the production line in a manufacturing workshop can be improved, which is significant for research on intelligent factories [3]. The intelligent factory takes the cyber–physical production system (CPPS) as the core and is characterized by the deep integration of information technology and manufacturing technology. It embeds the new generation of information technology, such as the Internet of Things, big data, and cloud computing, into different links of the manufacturing process to achieve the efficient production of intelligent products characterized by customization [4].

For smart factories, the introduction of intelligent equipment has made the processes within systems more convenient, but at the same time, the presence of a large number of devices such as sensors in the workshop will lead to the generation of massive amounts of data in real time, so it is still challenging to build a real-time data transmission management system. In addition, due to the large scale and relatively scattered production workshops of modern factories, and the fact that factories generally use intranets to control the entire workshop system to ensure security, the existing production collection system has limitations in processing big data. How to efficiently and securely integrate and manage this information has been a difficult problem faced by factories undergoing digital transformation in recent years [5].

The efficiency of traditional workshop production line data processing can no longer meet production needs. With the rapid development of technologies such as the Internet of Things and AI in recent years, the amount of data from edge devices in workshop production lines has doubled, which has brought huge challenges to the centralized big data processing mode of traditional cloud computing. Massive data need to be transmitted through the network to the computing center for centralized processing, resulting in inefficient data processing of edge devices, which reduces the real-time capabilities and scalability of the entire system [6].

Moshiri [7] pointed out that OPC UAover TSN (Time Sensitive Network), can support horizontal and vertical communication in smart factories, effectively reducing communication latency and improving system performance, providing a powerful solution for control-level communication in Industry 4.0. Trifonov [8] proposed an industrial IoT fieldbus solution based on OPC UA over TSN, and the experimental results verified the potential of OPC UA over TSN to achieve sub-millisecond latency in smart factory scenarios, meaning that it can meet the stringent requirements of smart factories in terms of the high real-time performance of industrial IoT. These studies show that the introduction of TSN technology significantly enhances the real-time performance of OPC UA in edge scenarios, but most of the studies focus on the communication level, and discussions of combination with intelligent inference or a data processing layer are limited.

Lai [9] and Takefusa [10] systematically evaluated Kafka, MQTT, and REST API in terms of latency, throughput, scalability, and replication performance. The results show that Kafka has advantages in partition scalability, throughput capacity, and ease of multi-node deployment, making it particularly suitable for real-time data processing needs in high-concurrency and distributed scenarios. Therefore, Kafka, with its strong scalability, natural support for partitioning mechanisms, and superior performance in large-scale distributed scenarios, has become an ideal choice for real-time data collection, transmission, and processing in edge computing environments.

Sharanya [11] studied the application of edge and fog computing in intelligent manufacturing, emphasized the importance of predictive maintenance as one of the core goals of Industry 4.0, and pointed out that edge AI technology has higher feasibility and application value to realize equipment condition monitoring and fault prediction in small- and medium-sized manufacturing enterprises. Han [12] studied the application of edge artificial intelligence (Edge-AI) in smart factories. He pointed out that it has significant advantages in real-time data collection, analysis, and prediction and can realize low-latency data transmission and risk warning between different devices. Li [13] proposed a smart factory reference framework that integrates digital twin and cloud-fog-edge collaborative computing (CFE), aiming to improve the efficiency of process data processing and enhance the generality and intelligent decision-making ability of the system. Singh [14] provides a comprehensive analysis of AI methods and functions related to edge computing or edge AI, opening up new possibilities for real-time decision making, data confidentiality, and system security in smart factories. Long [15] proposed the introduction of edge computing in the intelligent robot factory (iRobot-Factory), which effectively improves the production efficiency and automation level and has important reference value for the development of intelligent manufacturing. Although the above research confirms the broad prospects of Edge-AI in the manufacturing industry, there is still a lack of a systematic framework and verifiable practice cases on efficiently integrating edge-side inference capabilities with underlying reliable data communication architectures (such as Kafka and OPC UA) in actual deployment.

With the Industrial Internet of Things (IIoT) and cyber–physical systems (CPS) booming, manufacturing systems are moving toward higher levels of connectivity and intelligence. However, the current data management solutions have obvious shortcomings in communication bandwidth, processing capabilities for asynchronous data streams, and the collection and analysis of high-frequency heterogeneous data sources, which make it difficult to meet the increasingly complex data requirements in the new manufacturing environment. Despite the growing adoption of the OPC UA protocol as a standard for machine-to-machine communication, its default client–server model lacks an effective mechanism for large-scale real-time data streams. Similarly, traditional data-transfer protocols such as MQTT and RESTful APIs have limitations in handling high-throughput scenarios and ensuring strict real-time guarantees. This research aims to address these limitations by proposing a distributed real-time data management framework based on OPC UA and Apache Kafka and deploying it at the edge. Kafka's publish–subscribe architecture and persistence make it particularly suitable for high-concurrency and high-throughput industrial data transfers.

The research content of this paper is as follows:

1. Aiming at the difficulty of heterogeneous data collection in smart factories, a data-collection system based on the OPC UA protocol is developed. By modeling the workshop equipment with OPC UA or indirectly using PLC and other devices to communicate with the OPC UA client, real-time data collection of the underlying equipment is realized.
2. Based on edge computing technology, a data management framework for manufacturing production edge measurement is proposed. By arranging algorithm programs on the edge, the collected data are analyzed, processed, and then sent to the cloud to realize multi-platform sharing and storage of data. Compared with sending all data to the cloud, this not only relieves the pressure on the cloud but also improves the real-time data processing efficiency of the system.
3. The current big data transmission technology is analyzed, and a big data transmission framework for manufacturing workshops based on Kafka is designed. The Kafka

Topic is reasonably divided to improve the data transmission performance of the system. Finally, the database table structure is reasonably designed to optimize the data reading and storage speed and realize the persistent storage of manufacturing process data.

The remainder of this paper is organized as follows: Section 2 is a literature review, Section 3 introduces the proposed framework and research methodology, Section 4 discusses the experimental results, and Section 5 summarizes the thesis and outlines future research directions.

2. Review

With the development of intelligent industrial equipment, the importance of automatic detection and control is increasing, and the unified OPC UA architecture is rapidly popularized in modern intelligent manufacturing systems [16]. Due to the complex production process in the workshop, the various internal equipment and the difficulty of communication between the old and new equipment, many pieces of equipment have not been updated for a long time and do not support the new software, and the replacement is too expensive for the factory, so the OPC UA protocol is often used to solve the above problems. Ruben [17] proposed that OPC UA is a set of secure, reliable, manufacturer- and platform-independent data-exchange specifications for industrial communication, with a cross-platform service architecture, that enables communication between equipment and manufacturing floor systems. Ramesh [18] proposed a traditional device data integration system based on the OPC UA architecture, which can push data from the traditional PLC to the cloud to achieve remote data control.

In terms of data processing, smart factories can adopt a centralized structure based on traditional cloud computing and a decentralized structure based on edge computing. Cloud computing and edge computing both play a key role in the future development of the smart Internet of Things. The main differences between cloud computing and edge computing are shown in Table 1. The advantage of cloud computing architecture is that it stores large amounts of data on a central server, thus supporting large-scale data analysis using machine learning, data mining, and deep learning. However, it has the disadvantages of high cost and time latency due to the increase in network paths for transmitting data from the lowest layer to the highest layer. In the edge computing architecture, the cloud is deployed near the device layer, and data are not sent directly to the central server but processed at the middle layer, which reduces the communication path and input data transmitted to the central server [19]. Since edge computing is a structure suitable for manufacturing environments that require real-time data processing and rapid response, many studies have considered edge computing as an optimized platform technology for building smart factories.

Table 1. Main differences between cloud computing and edge computing.

	Applications	Network Bandwidth Pressure	Real-Time	Computing Mode
Cloud computing	Overall	More	High	Centralized processing at scale
Computing	Local	Less	Low	Small-scale centralized processing

When traditional factories are gradually upgraded to smart factories, due to the large amountof intelligent equipment, the high concurrency and high throughput of data in the manufacturing workshops lead to information congestion and delay in the entire collection system. The traditional system integrates data collection and processing. When processing large amounts of data (such as machine log data, sensor data, etc.), there are problems of poor versatility, insufficient flexibility, and scalability. How to efficiently transmit data and improve the system's rapid feedback capability is an urgent problem that the current manufacturing industry needs to solve. A distributed system (Cluster) is a promising method in big data research. The number of integrated server computers processes data in parallel, which reduces the coupling between the various parts of the system and reduces data redundancy. The main purpose of the distributed framework is to eliminate heterogeneous data by using a unified data transmission pipeline between the various components, thereby realizing data transmission and sharing of the entire framework.

As the information management center of the entire system, the message middleware mustbear a considerableamount of data exchange when facing massive data, so it still has deficiencies in throughput, delay, reliability, etc. With the continuous development of Internet technology, distributed message middleware came into being to solve the shortcomings of traditional message middleware in these scenarios. Different from traditional message middleware, distributed message middleware has significant advantages in data bandwidth, processing performance, reliability, fault tolerance, etc. due to its distributed architecture, enabling it to efficiently and reliably handle massive message data transmission. Its main advantages are as follows:

1. Entity decoupling: Producers and consumers do not need to know each other; they only need to communicate and interact with the message middleware.
2. Time decoupling: Producers and consumers do not need to be online simultaneously to participate in the communication.
3. Synchronous decoupling: The producer or consumer publish/subscribe infrastructure does not need to synchronously block the creator or consumer execution thread, allowing the maximum utilization of the processor resources of the creator and consumer.

Apache Kafka is a popular open-source distributed stream processing platform widely used to build network data flow platforms [20]. As a publish/subscribe system for network data flow, Apache Kafka provides a highly scalable and fault-tolerant solution with the potential to easily process large amounts of data [21]. Sangil [22] proposed a data collection and stable exchange system based on Apache Kafka to address the problems of production bases scattered around the world and information exchange difficulties. Hesse [23] and Atri [24] conducted a comprehensive performance analysis of the Apache Kakfa platform. The experimental results show that it has scalability, distribution, and reliability with high throughput.

3. Materials and Methods

In the context of the rapid development of intelligent manufacturing, the data generated on industrial sites present the characteristics of large data volume, multiple types, and strong real-time performance, which puts forward higher requirements for data collection, processing, and transmission capabilities. At present, industrial communication protocols such as OPC UA have been widely used in the data collection of underlying devices; however, there are still challenges in multi-source heterogeneous device access, communication delay, and data integration. The traditional cloud computing model has limitations in real-time and bandwidth utilization. The introduction of edge computing effectively alleviates these problems, realizes local processing and rapid response of data, and improves the stability and intelligence level of the system. However, edge devices still need to be further optimized in terms of computing power, protocol adaptation, and collaboration with the cloud. Therefore, this study designs and implements an edge data collection and transmission framework based on OPC UA and Kafka around the high-concurrency data collection and processing requirements in intelligent manufacturing scenarios to improve the system's data processing capabilities and response efficiency and provide support for the efficient use of industrial big data.

As shown in Figure 1, the architecture includes a data collection layer, a data processing layer, a data transmission layer, and a data application layer, which is mainly composed of PDC (Production Data Collection) components, edge-computing modules, Kafka clusters, databases, and Web front ends. At the edge node layer, the OPC UA protocol is used to collect equipment production data in a standardized manner. The edge-computing algorithm is used to process and analyze the raw data, and the results are forwarded to the distributed Kafka service cluster for persistent storage. At the storage layer, PostgreSQL is used to manage user and equipment metadata, and the Kafka cluster is used to store a large amount of process data produced by equipment to achieve scalable and reliable storage of large-scale data and to share key data through the cloud.

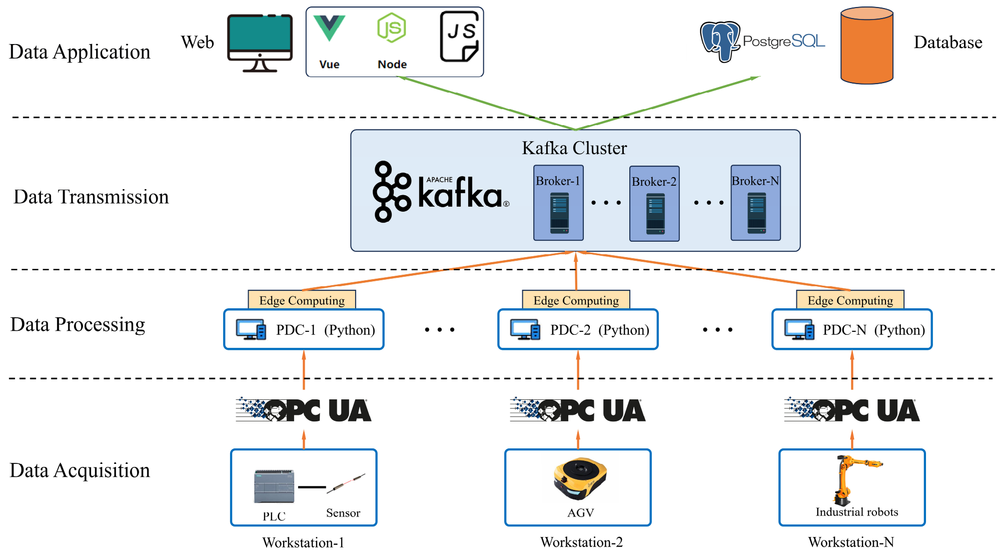


Figure 1. Real-time data acquisition and processing architecture based on the OPC UA protocol.

3.1. Design of Data Acquisition Module Based on OPC UA Protocol

3.1.1. Building an OPC UA Server

In this article, the custom-written OPC UA Server and OPC UA Client are deployed on the same PC (computer) and are responsible for the data collection of the devices on this unit. Therefore, the OPC UA server is used to realize the object modeling and construction of the internal equipment of the production workshop, and the OPC UA server can model the internal equipment of the production workshop and expose the equipment data for users to access in a unified manner.

The design idea of the data acquisition module based on the OPC UA protocol is as follows: firstly, it is classified according to the communication protocol of all devices in the production workshop, and the equipment with built-in OPC UA server or the equipment that can communicate indirectly through the PLC node can directly use the customized OPC UA client to communicate with it. Suppose the device does not support the OPC UA server. In this case, the device is modeled using OPC UA, and the OPC UA node is mapped one by one to achieve unified OPC UA communication across the entire manufacturing workshop, as shown in Figure 2.

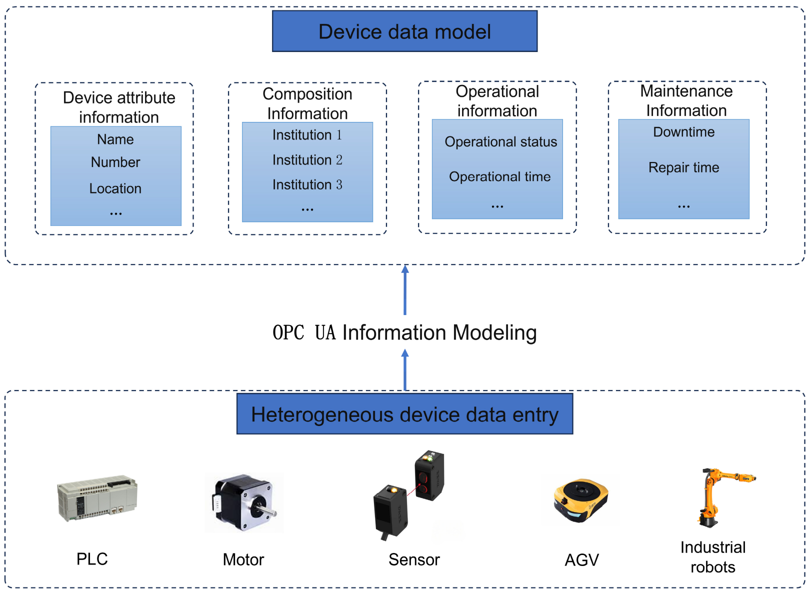


Figure 2. OPC UA information modeling.

The data collection layer includes an OPC UA server and the data collection program. The OPC UA communication module can collect data, equipment status information, and actual processing status information of the bottom-level equipment in the manufacturing workshop, such as PLCs, sensors, and motors, and send them to the upper layer of the system. The OPC UA server module establishes a node address space model for those devices that do not support the OPC UA protocol, thereby realizing unified communication of the entire manufacturing workshop based on the OPC UA protocol. The code is shown in Appendix A. The specific development process is as follows:

1. Define the device type. Before creating an OPC UA object for the device, the device type must be defined.
2. Define the device object. According to the previous device object model, encapsulate the device object model, reference the device and the parameters and methods related to the device by node, and then create a namespace for the device so that users can access the target device through this address space.
3. Bind real-time data, connect to the OPC UA server, bind the device parameters to the data tags in the OPC UA server, and provide real-time data for the device parameters. Develop a read–write interface and a historical read interface. For other systems to obtain device data, a data access interface needs to be developed.
4. Develop access interfaces, including read/write interfaces and historical read interfaces. Through these interfaces, users can browse the device's specific parameters.
5. Create a public data access address. By configuring the OPC UA server, specify the public address for data access. This address is used for other systems to establish communication connections with the data center.
6. Data storage management. In order to extract more value from device parameter information, the device's real-time data needs to be saved.

3.1.2. Data Acquisition Module Based on the Python Language

With the widespread introduction of intelligent equipment in smart factories, the number of data sources, including CNC machine tools, industrial robots, programmable logic controllers (PLCs), automatic guided vehicles (AGVs), and various sensors, has increased significantly. The diversity of communication protocols and interfaces used by these devices has led to the demand for customized communication solutions, which not only increases the complexity of data collection but also poses a significant challenge to the production efficiency and development of smart manufacturing workshops. Therefore, this study proposes a distributed framework that uses different PDC components for devices with different communication protocols to achieve distributed data collection of underlying devices in the workshop, thereby improving the efficiency of the data collection of the entire workshop equipment, as shown in Figure 3.

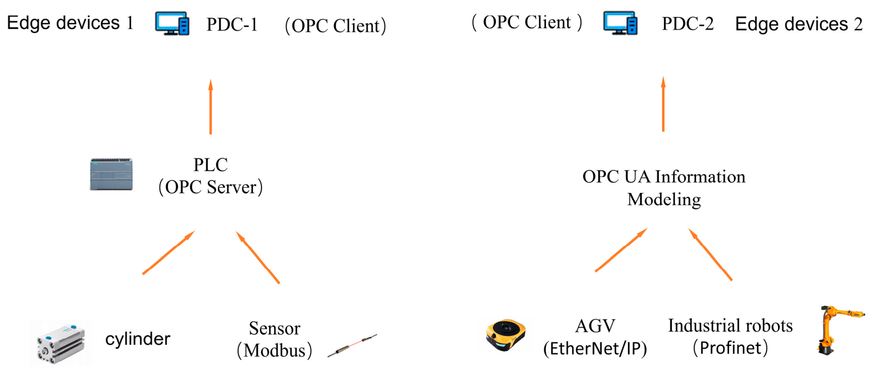


Figure 3. Distributed data acquisition architecture diagram based on the OPC UA protocol.

This article uses the OPC UA library to design a PDC component, which integrates the OPC UA client function and can communicate with the equipment in the manufacturing workshop through the URL address. The design process of the PDC component mainly includes the following steps:

1. Start the Kafka service locally. Then, create a new topic named "query" and subscribe to it to receive the list of node identifiers provided by the database for monitoring. At this stage, the database side acts as the Producer of Kafka, sending the data information to be monitored pre-written by the workshop, and the PDC side receives this information as the Consumer of Kafka.
2. When PDC can pull data from "query", it will process the data, filter out the URL address of the PLC to be detected and all nodes under the address, and integrate the above information into the form of an object in Python. It will then traverse and read all objects, connect to the OPC UA server through the URL, process the node to be detected, obtain the address space index of the node, and then integrate it into a list with NS (NamespaceIndex) and I (Identifier) values.
3. Subscribe to all the node states in the above list through the subscription function. When the value of the node changes, the sending function in the PDC component will be called. At this time, PDC plays the role of Kafka Producer and sends the message to the corresponding topic.

3.2. Production Monitoring and Inventory Management Based on Edge Computing

In the previous chapter, the data collection of the underlying equipment in the manufacturing workshop was realized, and if all the equipment data information collected is sent to the cloud for unified processing, it will create greater computing pressure in the cloud system, and at the same time, it is impossible to ensure the real-time monitoring and control of the entire system equipment, so the framework of rule reasoning (RBR)—edge computing (edge computing) suitable for industrial production lines is proposed, which can quickly judge the production status and update the inventory locally without complex calculations. This is shown in Figure 4. Using the OPC UA client as the edge computing device, an edge-computing program at the edge end is designed, which initially analyzes and processes the collected data to realize station status identification and inventory update. Finally, the results are sent to the cloud in a unified manner, which reduces the pressure on the cloud server, improves the computing power and transmission capacity of the server, and ensures the real-time data of the entire system. Even if the network connection is disconnected, the edge can ensure the normal operation of the real-time data acquisition program without downtime, reduce the data flow during transmission, and ensure the security of plant data.

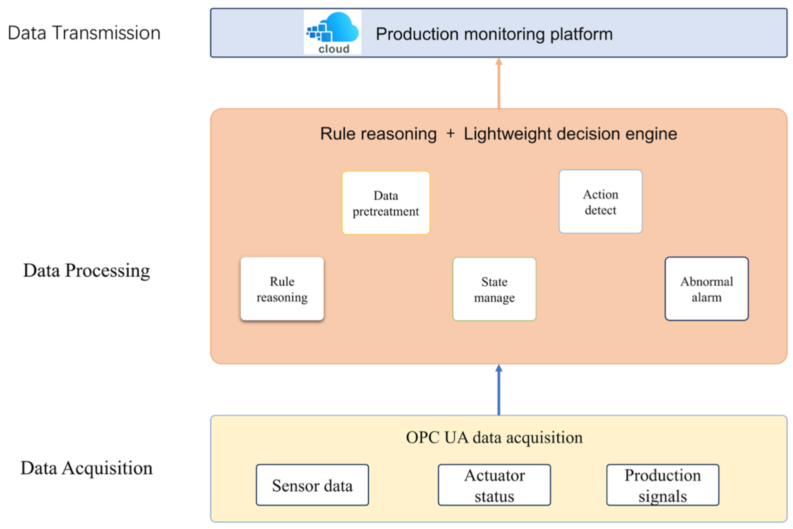


Figure 4. Diagram of data processing architecture based on edge computing.

The actuator equipment that needs to be detected by the platform includes stepper motors, sensors, cylinders, etc. However, these devices themselves do not support the OPC UA communication protocol, so the state data of the actuator can be obtained indirectly with the help of an intermediate relay when the PLC program diagram of the experimental platform is designed to realize the data acquisition work of the entire experimental platform. Among them, the experimental platform is divided into two categories according to the information to be detected: one is the action of feeding, which needs to update the inventory information in real time, and the other is the action of installation, which needs to monitor the production progress of the production process in real time. The analysis of the production process found that the experimental platform has a total of three stations related to materials, namely, the device for pushing the bottom (BM), the feeding device (KM) for releasing beads, and the device for pushing the upper cover (DM), so this study uses the intermediate relay of PLC to obtain data. As shown in Figure 5, when designing the program of the feeding device (KM) of the second station to release the beads, the original design idea is to control the power-on and power-off of the enabling signal of the pulse signal through the timer switch. This study adds an intermediate relay %M1.0 set program block in the middle of the timer and the pulse module. When the button switch KM is pressed, the intermediate relay %M1.0 will be set to 1, and the enable signal of the pulse generator will also be set to True. When the timer is disconnected, the intermediate relay % M1.0 reset is 0; at the same time, the pulse generator will also be powered off. Therefore, the intermediate relay %M1.0 and the stepper motor pulse for on and off is the same. By detecting the intermediate relay %M1.0 to obtain the state of the stepper motor, real-time data acquisition and monitoring objectives of the station can be achieved, and at the same time, it can reflect the material inventory data by the number of times the station action is completed. The device (BM) at the bottom of the station and the device (DM) on the top of the station push can also realize real-time data collection and inventory monitoring in this way.

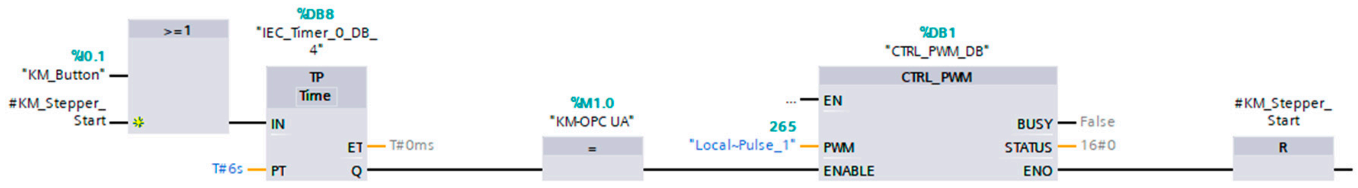


Figure 5. PLC programming of the loading device (KM) to release the beads.

According to the above method, by detecting the data changes of the specified nodes, data collection and inventory monitoring can be realized. Therefore, this article uses Python language to write a real-time data monitoring program for the manufacturing workshop based on the edge data collection terminal PC. By analyzing the functions of different workstations and designing corresponding data processing and analysis programs, the entire production process and inventory information can be comprehensively judged.

3.3. *Stream Processing and Storage of Manufacturing Data*

3.3.1. Kafka System Overall Design

The data stream processing framework of this article is shown in Figure 6. All servers used for data collection, storage, or visualization are organized into a Kafkacluster. Each edge server that integrates the PDC component acts as a Kafka producer responsible for data collection and message sending in different manufacturing units.

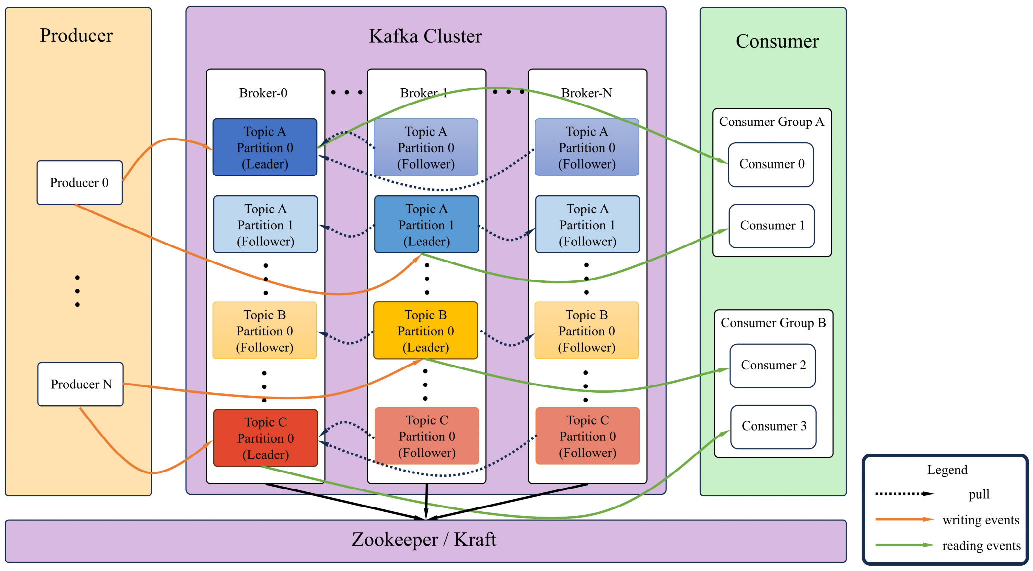

Producer
Producer 0
Producer N
Kafka Cluster
Broker-0
Broker-1
Broker-N
Topic A Partition 0 (Leader)
Topic A Partition 0 (Follower)
Topic A Partition 1 (Follower)
Topic A Partition 1 (Leader)
Topic B Partition 0 (Follower)
Topic B Partition 0 (Leader)
Topic C Partition 0 (Leader)
Topic C Partition 0 (Follower)
Consumer
Consumer Group A
Consumer 0
Consumer 1
Consumer Group B
Consumer 2
Consumer 3
Zookeeper / Kraft
Legend
pull
writing events
reading events

Figure 6. Apache Kafka architecture diagram.

The PostgreSQL database and front-end Web components responsible for data storage and backup serve as Kafka's Consumer Group A and Consumer Group B consumer groups and realize message consumption by subscribing to the specified Topic. Since Kafka stores messages in Broker, message sending and consumption are directly interacted with Broker, so to a certain extent, the decoupling of the message transmission process is achieved—that is, the producer and consumer are completely decoupled, ensuring that data are not lost, with good fault tolerance, and realizing the persistent storage of messages and distributed multi-platform shared consumption.

3.3.2. Kafka Topic Design

Due to the large scale and wide distribution of modern production workshops, the amount of data generated is huge. In addition, the increase in intelligent equipment has introduced significant challenges to the digital monitoring and management of workshops. Topics in Kafka can be used to divide the source of messages. Traditional data-collection systems store all collected data in a Topic, and then consumers pull messages according to the offset of the message data; second, any possible message data type is processed by the Topic. Consumers only need to subscribe to the specified Topic without all data being consumed uniformly, and finally analyzed and screened. Both of the above methods put pressure on network bandwidth or computing performance, so they are more suitable for scenarios with simple data formats and small data volumes. Unreasonable Topic division has a negative impact on the scalability of the system, and the number of Topics directly affects Kafka performance and maintenance costs. Therefore, it is necessary to comprehensively consider the needs of the manufacturing workshop and consider selecting a suitable Topic division strategy to optimize the design based on factors such as performance overhead, cost expenditure, and message volume.

This paper proposes a new KafkaTopic-based partition strategy to divide different types of data on the manufacturing floor into different topics, as shown in Figure 7. Each topic manages the messages of all nodes under the collection framework, which not only divides the workshop data and facilitates the tracking and management of messages but also improves the efficiency of the entire system in consuming messages.

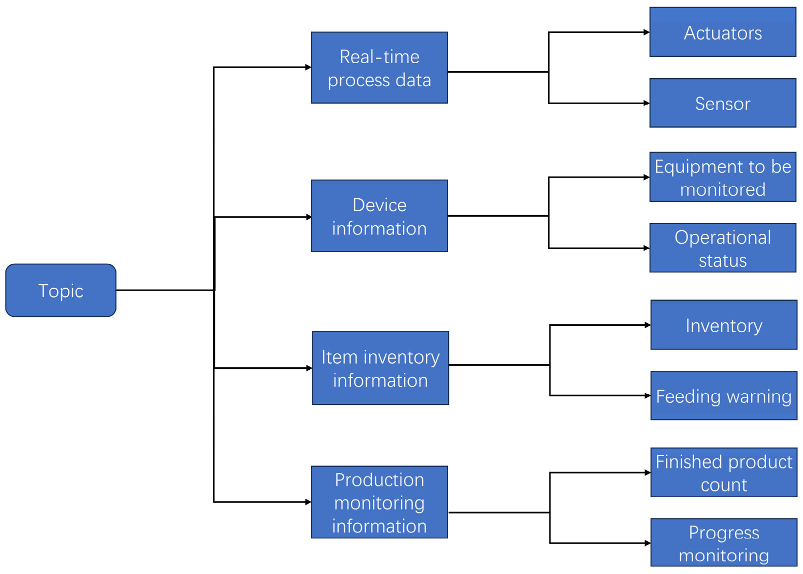


Figure 7. Kafka Topic partitioning logic.

3.3.3. Message Sending and Receiving Module Development

The message sending and receiving module encapsulates the ProducerAPI and ConsumerAPI provided by Kafka into a class in Python. When the value of the subscribed node changes, the program calls the class by definition and uses the producer method to send the message to the specified Topic, which is automatically appended to the end of the Topic. The sending and consumption example is shown in Figure 8. The figure consists of a data collection module and a Topic with three partitions: P1, P2, and P3. When a new message is sent, the Kafka Producer will split the message according to the number of partitions defined by the user and send it to the specified partition for storage. At the same time, Kraft will record the offset of this message in the partition. When a Kafka Consumer wants to consume the data, it can find the offset of the target data in Kraft and retrieve the data from the specified partition. At the same time, Kafka also supports multiple consumer modules in consuming the same data at the same time, thereby speeding up the data-consumption process and improving the real-time response speed of the system.

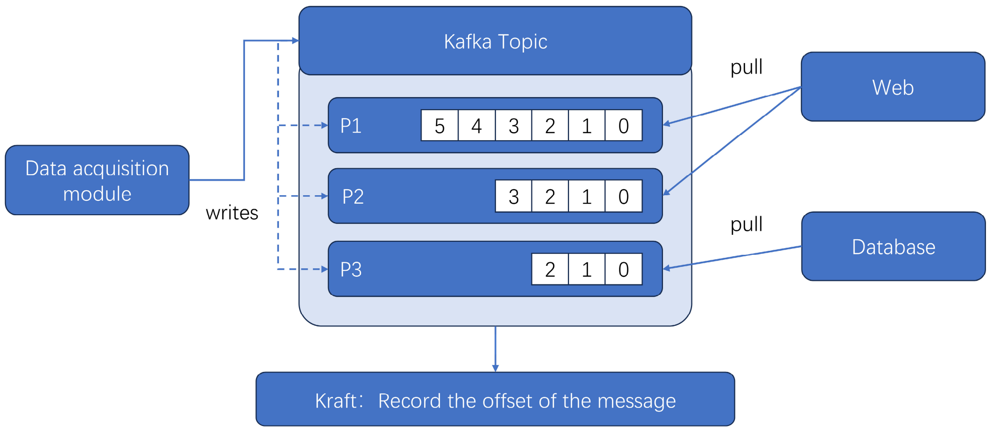


Figure 8. Example of sending and consuming Kafka messages.

3.3.4. Multithreaded Parallel Data Consumption

With the continuous development of computer technology, multi-threading technology has been gradually adopted by smart factories, the main purpose of which is to reduce the idle time of the central control computer of the smart factory, give full play to its performance so as to enhance the throughput of the whole system, and improve the response speed and overall performance of the data acquisition end and the application side. However, in a large number of devices in a smart factory, the individual acquisition threads do not continuously send and receive data but are only activated when the data state changes. As a result of this feature, threads are frequently created and destroyed, which puts greater pressure on system resources and reduces processing efficiency.

For this reason, this paper uses a custom implementation of a thread pooling mechanism based on Python's threading module to process the real-time data obtained from Kafka for multi-threaded consumption, as shown in Figure 9. Thread pools avoid the overhead of frequent thread creation and destruction by pre-creating a fixed number of Worker Threads that pull pending Kafka messages from a shared task queue. In the specific implementation, the thread pool is configured with the following parameters:

1. Thread pool size (number of worker threads): set to 8, according to the number of CPU cores (4 cores and 8 threads) of the experimental server, to achieve a balance between concurrency and system resource occupation;
2. Task queue size: set to 1000 to cache the pending messages consumed by Kafka to avoid thread blockage caused by burst data.
3. Blocking policy: When the task queue is full, the producer thread (Kafka consumer) blocks the waiting queue to have empty seats to ensure the stable operation of the system.
4. Thread lifecycle management: All threads start when the main program is initialized and continue to cycle through the task until the program exits.

This mechanism realizes the quasi-real-time concurrent processing of Kafka messages in actual operation, which has good throughput and system responsiveness and avoids the additional overhead caused by frequent thread creation. The thread pool size and queue capacity can be flexibly adjusted based on Kafka message throughput and server resources.

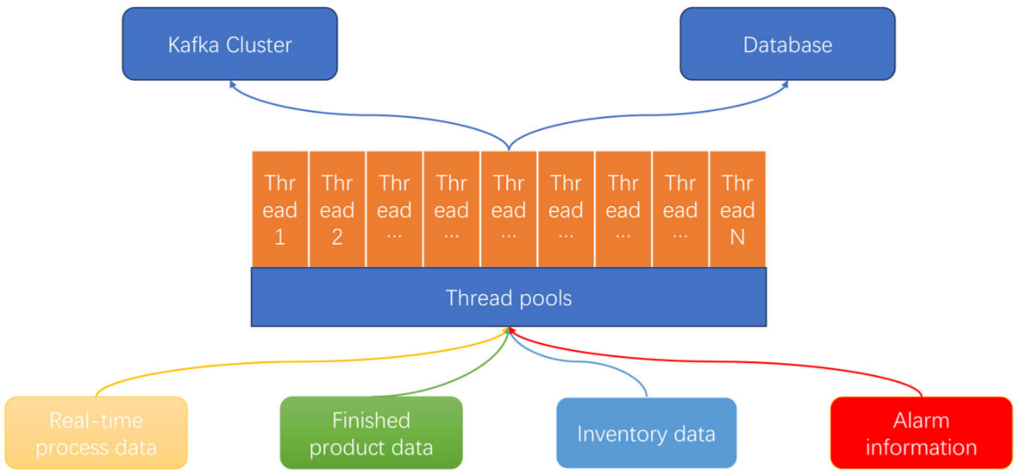


Figure 9. Multithreaded parallel consumption.

3.4. *Design of Storage Scheme for Manufacturing Data*

With the transformation of traditional factories to intelligent ones, the amount of data in factories has exploded, and the demand for real-time data processing has increased. It is particularly important to choose a suitable database optimization strategy. A reasonable design of the structure of the database table can improve the performance and efficiency of the database, reduce the association operation of the table, and improve the system response speed. On the other hand, different data types can be selected according to the characteristics of different data, and the use of overly large or small data types can be minimized to reduce the waste of storage space and improve query efficiency. At the same time, appropriate primary keys and foreign keys can be selected for each table to ensure the integrity and consistency of the data and avoid using too many or too long primary keys and foreign keys to reduce the storage and maintenance costs of indexes.

This study is designed to support the data collection architecture of the entire production workshop, because there are many devices in the production workshop. Equipment with the same specifications will be distributed in different stations in different production and manufacturing units. The information of each equipment to be detected includes the equipment number, node ID, UUID, station ID (PLC_ID), URL, manufacturing unit ID, name, and location, etc. Therefore, this study uses the information of the whole workshop to split into a number of smaller tables for decentralized storage, carries out a reasonable table sharding design according to the characteristics of the data, and divides each equipment to be detected into three tables of different dimensions for storage and classification, which is sorted in a tree shape, as shown in Figure 10.

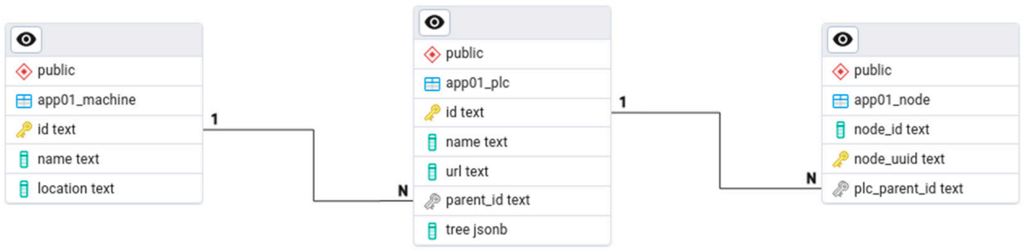


Figure 10. E-R diagram of database tables.

The bottom-level table named app01_node contains some basic information of the device, such as the node value and the UUID sequence that uniquely identifies the device. The middle-level table app01_plc contains the station information of the device, such as the station number and the URL for the OPC UA client connection. The top-level table app01_machine contains the name, number, and location of the manufacturing unit to which the station belongs. Tables app01_machine and app01_plc select the id column as the primary key. Tables app01_plc and app01_node select the parent column as the foreign key and then associate the foreign key with the primary key of the previous layer so as to associate all tables together to reduce the amount of data and the number of indexes of a single table, thereby improving query efficiency.

The database side also integrates the program responsible for processing and writing message data into the table, covering the following key links:

1. Start Kafka on the local database side, then connect to the database and query the device information in a table, and obtain the node information to be detected, including the node ID, the unique identifier UUID, the PLC_ID corresponding to the node, the connection address URL, and the machine ID.
2. Process and classify each of the above information, divide them by URL, integrate the nodes under the same connection address into the form of a dictionary in Python, optimize the data structure, and improve the PDC retrieval efficiency.
3. Convert the above integrated messages into JSON and send them to the Topic named "query". At the same time, different Topics will be created based on different PLC_IDs to optimize the production and consumption rate of messages.
4. Subscribe to Topics with different PLC_IDs. When consuming new messages, the current time will be automatically obtained, and then the current time will be written together with the message into the specified data table to achieve storage and backup of factory production data.

4. Experimental Testing

4.1. Design and Construction of Assembly Experimental Platform

The experimental object of this paper is an automatic assembly platform composed of six stations, including automatic loading, transfer, assembly, and handling. First of all, the entire platform is allocated to stations according to function and processing sequence, and each station has a specific execution to complete the set action. The design of the platform mainly includes mechanical structure design, electrical component selection, electrical schematic design, and PLC program writing and debugging. The overall structural layout is shown in Figure 11.

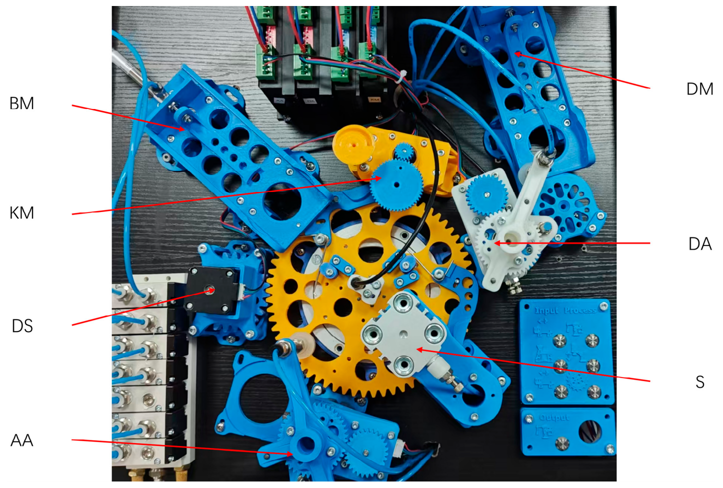


Figure 11. The overall layout of the assembly experimental platform.

The mechanical structure of the entire platform was drawn in Solidworks, 3D printed using PLA material, and then assembled with electrical components. It mainly includes six work areas, namely, the device responsible for pushing the bottom (BM), the loading device for releasing beads (KM), the device for pushing the upper cover (DM), the rotating device for transporting the upper cover (DA), the clamping device (S), the device for transporting and shipping (AA), and the rotating device responsible for the bottom platform (DS).

4.2. Test Environment Configuration

In order to simulate the working conditions of information collection in a real manufacturing workshop and verify the feasibility of the entire framework, this paper uses a server as the main test machine and uses VMware software to simulate three different servers as three Kafka Brokers to form a Kafka cluster. The configuration of each simulated server is shown in Table 2. All use a unified Ubuntu 18.04 as the operating system and use Kraft for unified management of the Kafka cluster.

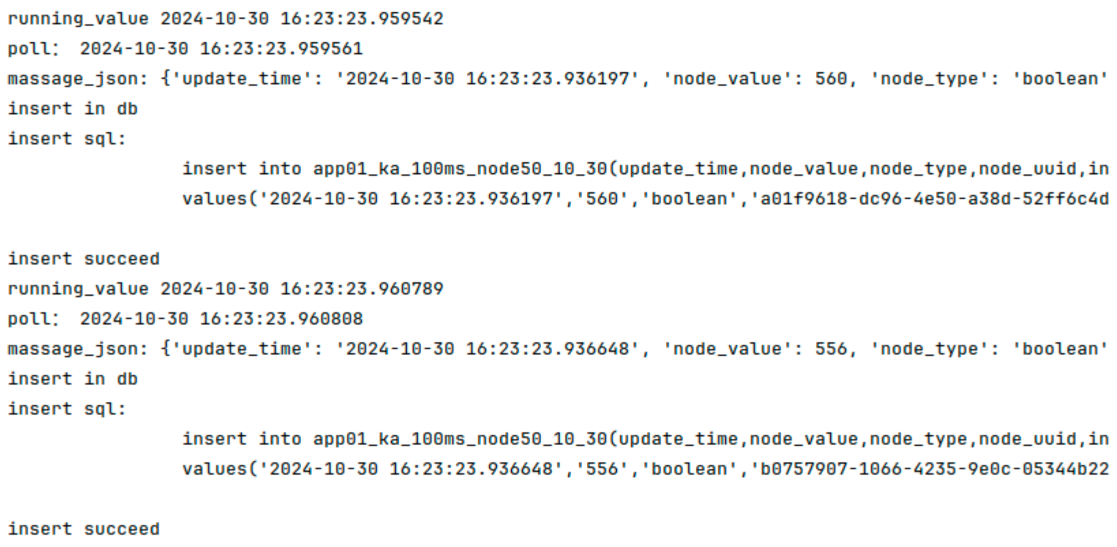
Table 2. Test configuration of the server.

CPU	RAM	HDD	OS	Kafka Version	PostgreSql Version
4 × 3	16 GB	50 GB	Ubuntu22.04.4 LTS	2.13–3.7.0	16.3

CPU: Central Processing Unit. RAM: Random Access Memory. HDD: Hard Disk Drive. OS: Operating System.

4.3. Manufacturing Workshop Data Collection Terminal Test

This article uses three servers to build a test platform. Two of the servers integrate PDC components as OPC UA clients, namely, PDC1 and PDC2. Each PDC is responsible for detecting the underlying equipment data under the manufacturing unit. The other server uses PostgreSql as the data storage function, which is responsible for pulling the data collected by the two PDC servers from the Kafka Cluster. The Web front-end component realizes data visualization by subscribing to the specified Topic. The entire framework uses JSON format for data exchange, and each piece of data has a uniquely identified UUID sequence defined to facilitate the distinction of a large number of devices in the workshop, as shown in Figure 12. Each piece of collected data contains the data change time, variable value, data type, and node identifier UUID of the node.



```
running_value 2024-10-30 16:23:23.959542
poll:  2024-10-30 16:23:23.959561
massage_json: {'update_time': '2024-10-30 16:23:23.936197', 'node_value': 560, 'node_type': 'boolean'
insert in db
insert sql:
                insert into app01_ka_100ms_node50_10_30(update_time,node_value,node_type,node_uuid,in
                values('2024-10-30 16:23:23.936197','560','boolean','a01f9618-dc96-4e50-a38d-52ff6c4d

insert succeed
running_value 2024-10-30 16:23:23.960789
poll:  2024-10-30 16:23:23.960808
massage_json: {'update_time': '2024-10-30 16:23:23.936648', 'node_value': 556, 'node_type': 'boolean'
insert in db
insert sql:
                insert into app01_ka_100ms_node50_10_30(update_time,node_value,node_type,node_uuid,in
                values('2024-10-30 16:23:23.936648','556','boolean','b0757907-1066-4235-9e0c-05344b22

insert succeed
```

Figure 12. Sample data received from the database.

In a smart factory environment, large amounts of data from OPC UA device nodes need to be frequently collected and transmitted. In order to achieve good cross-platform compatibility and readability, this study uses JSON format as the serialization method for Kafka messages. However, JSON itself has the problems of redundant field names and large encoding volume, which may cause a waste of transmission bandwidth and processing delay in high-frequency data scenarios. Therefore, compressing the message size and improving the transmission efficiency of Kafka while maintaining the flexibility of the system are key issues to consider.

In order to improve the efficiency of message transmission, this study tests and compares three mainstream compression algorithms: gzip, lz4, and snappy, with the help of the message compression mechanism natively supported by Kafka. The experimental results are shown in Table 3.

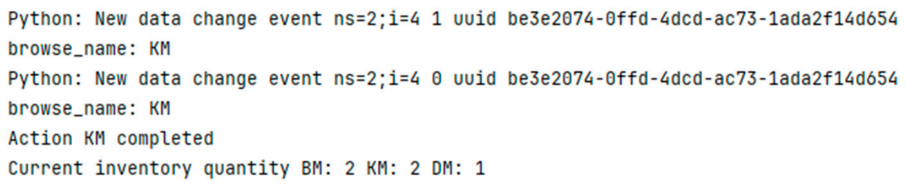
Table 3. Comparison of JSON message compression algorithms.

Compression Algorithm	Average Compression	Ratio Compression/Decompression Time	Applicable Scenarios
Gzip	70–75%	high	High compression ratio
Lz4	55–60%	low	Real-time
Snappy	50–55%	medium	General scenes

Since the average size of a single JSON message required for transmission in this paper is 520~600 bytes, lz4 is finally selected as the Kafka message compression algorithm based on the compression effect and real-time processing performance. The algorithm has an extremely fast compression and decompression speed while maintaining a high compression ratio, which is suitable for the latency-sensitive real-time data processing requirements of smart factories.

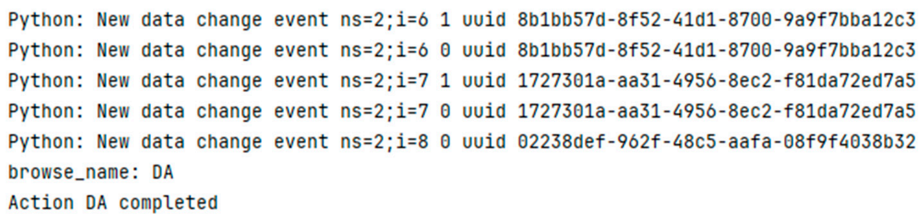
4.4. Manufacturing Workshop Edge Data Processing Test

In order to test the feasibility of the edge-computing method proposed in this paper in actual manufacturing production, this section conducts real-time data collection and analysis in the actual production process of the assembly experimental platform, and the results are shown in Figures 13 and 14. In the results, it can be seen that when the data acquisition end detects that the equipment data have changed, its data will be aggregated into the edge-computing program, and through the logical analysis and judgment of the program, real-time monitoring of the production process and inventory management can be achieved.



```
Python: New data change event ns=2;i=4 1 uuid be3e2074-0ffd-4dcd-ac73-1ada2f14d654
browse_name: KM
Python: New data change event ns=2;i=4 0 uuid be3e2074-0ffd-4dcd-ac73-1ada2f14d654
browse_name: KM
Action KM completed
Current inventory quantity BM: 2 KM: 2 DM: 1
```

Figure 13. Example of inventory calculation.



```
Python: New data change event ns=2;i=6 1 uuid 8b1bb57d-8f52-41d1-8700-9a9f7bba12c3
Python: New data change event ns=2;i=6 0 uuid 8b1bb57d-8f52-41d1-8700-9a9f7bba12c3
Python: New data change event ns=2;i=7 1 uuid 1727301a-aa31-4956-8ec2-f81da72ed7a5
Python: New data change event ns=2;i=7 0 uuid 1727301a-aa31-4956-8ec2-f81da72ed7a5
Python: New data change event ns=2;i=8 0 uuid 02238def-962f-48c5-aafa-08f9f4038b32
browse_name: DA
Action DA completed
```

Figure 14. An example of a production action calculation.

4.5. Manufacturing Workshop Data Transmission Test

In order to test the throughput of the system, the OPC UA server customized in Python was used to simulate the data changes of equipment in a real production workshop. The production and consumption of 10,000, 30,000, and 50,000 data were tested, respectively. lz4 format was used to compress the messages. The six groups of test results are shown in Figure 15. The results show that 99.9% of the message sending delays are within 1 s, and the average system throughput is 51.96 MB/s.

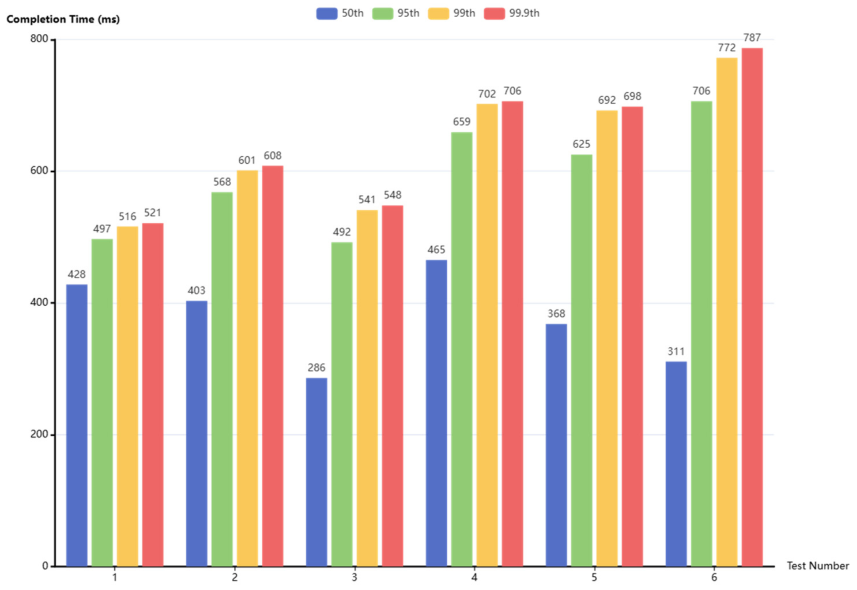


Figure 15. System throughput test.

In order to evaluate the performance of the entire framework in real-time data collection and processing, study paper uses Python to design an OPC UA server to simulate the generation of real workshop big data; change the value of the node at a frequency of 100 ms; and conduct actual tests on data collection, processing, and transmission based on the Kafka-based distributed processing framework. Given that Kafka uses batches to send and consume data, this study conducted seven sets of data sending and receiving tests, each containing about 50 messages. Figure 16 shows the delay from the sender to the receiver of each set of data.

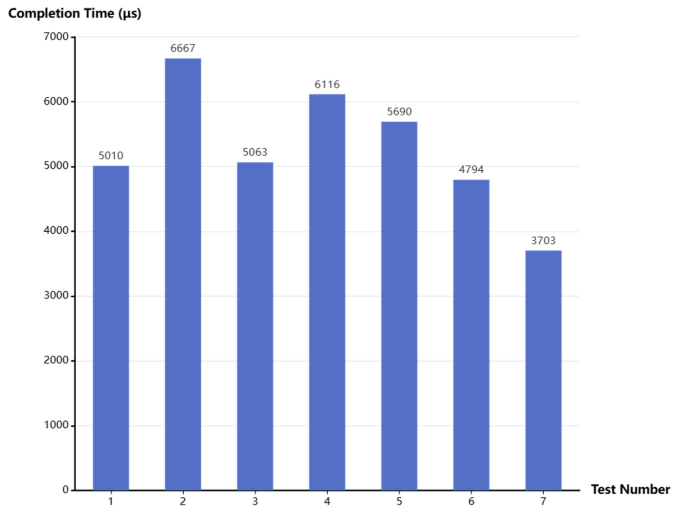


Figure 16. Data latency analysis after Kafka is introduced.

When the PDC program detects that the data of the specified node has changed, the system will specify the current time as update_time. When the database side pulls data from the Kafka Cluster and writes it to the database, the system will also specify the current time as insert_time. This article uses two different times of the same data as two time nodes for delay comparison. The results show that all data are completed from data collection to data storage within 7 ms, and the introduction of Kafka improves the throughput and horizontal expansion capabilities of the entire system and has a wider range of applications.

5. Conclusions

5.1. Summary

This study mainly uses industrial Internet of Things technology, big data processing technology, and edge computing to realize real-time detection, processing, and analysis of production data, equipment status, and production progress of intelligent production lines. The underlying system uses the OPC UA protocol to realize the standardized collection of heterogeneous equipment data, and the edge side performs preliminary analysis and processing of key data, which improves the real-time performance and response efficiency of data processing. The processed data are efficiently transmitted through the Kafka message queue and are persistently stored and backed up by the PostgreSQL database. Finally, the real-time monitoring and tracking of the entire production process is realized through the front-end visual interface, providing data support for production process optimization and decision-making.

The current testbed deploys Kafka nodes based on virtual machines (VMs), which can simulate the industrial network environment to a certain extent, but does not fully reflect the complexities of real industrial deployments, such as equipment performance differences, network congestion, and system heterogeneity. Therefore, future research will further deploy this framework in actual industrial sites to verify its stability and practicability in complex environments.

5.2. Implications

Although the distributed real-time data management framework proposed in this study shows good performance and scalability in the test, it still has the following limitations:

1. At present, the system has not yet integrated a real-time anomaly detection mechanism, and in the future, a lightweight model can be introduced at the edge node to achieve rapid response. Second, although the framework supports integration with edge AI, it lacks in-depth research on edge inference and model deployment. Further research can focus on the adaptability of federated learning or model compression technologies in practical applications.
2. Although a TSN-based OPC UA communication scheme is designed in this study, its performance in very low-latency scenarios (such as multi-device cooperative control) has not been fully verified, and it should be verified through more complex industrial experiments in the future.
3. In terms of security, the current solution mainly relies on the built-in mechanism of Kafka, and in the future, the system should introduce a zero-trust architecture, end-to-end encryption, and edge privacy protection policies to meet the multi-node deployment and data security requirements in the industrial Internet.

This study provides the academic community with the implementation idea of a system integrating OPC UA, Kafka, and edge computing and proposes a reference architecture for data processing in intelligent manufacturing. For the industrial community, the framework can be applied to key scenarios such as equipment condition monitoring and predictive maintenance and provides a foundation for building low-latency and highly reliable edge intelligent systems in the future. In the future, it can continue to expand to edge AI integration, real-time anomaly detection, OPC UA TSN key control scenarios, and industrial data security protection mechanisms.

Author Contributions: Conceptualization, D.L.; methodology, D.L. and K.W.; software, K.W.; validation, Y.W.; formal analysis, Y.W. and Y.S.; investigation, D.L. and K.W.; resources, Y.W.; data curation, Y.S.; writing—original draft preparation, K.W.; writing—review and editing, D.L. and Y.W.; visualization, K.W. and Y.S.; supervision, Y.S.; project administration, Y.W.; funding acquisition, D.L. All authors have read and agreed to the published version of the manuscript.

Funding: This work was supported by funding from the Shanghai Committee of Science and Technology (21010504000).

Institutional Review Board Statement: Not applicable.

Informed Consent Statement: Not applicable.

Data Availability Statement: The original contributions presented in this study are included in the article. Further inquiries can be directed to the corresponding author.

Acknowledgments: The authors would like to express their sincere gratitude to the Shanghai Institute of Technology.

Conflicts of Interest: The authors declare no conflicts of interest.

Appendix A

```
def create_server(endpoint, name, namespace_uri, node_dict):
server = Server()
server.set_endpoint(endpoint)
server.set_server_name(name)

idx = server.register_namespace(namespace_uri)

myobj = server.nodes.objects.add_object(idx, “MyObject”)
variables = []

for var_name, (ns, i) in node_dict.items():
print(‘正在创建变量:’, var_name, ns, i)
var = myobj.add_variable(ns, var_name, 0.0, ua.VariantType.Float)
var.set_writable()
variables.append(var)
print(‘已创建变量列表:’, variables)

server.start()
print(f”服务器已启动, 访问地址: {endpoint}”)
return server, variables
```

```
if __name__ == "__main__":
logging.basicConfig(level=logging.WARN)
server1_endpoint = "opc.tcp://0.0.0.0:4843/freeopcua/server1/"
server1_name = "Server 1"
server1_namespace_uri = "http://examples.freeopcua.github.io/server1"
server1_node_dict = {
"DS": (2, 2),
"BM": (2, 3),
"KM": (2, 4),
"DM": (2, 5),
"DA-S": (2, 6),
"DA-C": (2, 7),
"DA-V": (2, 8),
"S": (2, 9),
}
server2_endpoint = "opc.tcp://0.0.0.0:4844/freeopcua/server2/"
server2_name = "Server 2"
server2_namespace_uri = "http://examples.freeopcua.github.io/server2"
server2_node_dict = {
"AA-S": (3, 1),
"AA-C": (3, 2),
"AA-V": (3, 3),
}
server1, server1_vars = create_server(
server1_endpoint, server1_name, server1_namespace_uri, server1_node_dict
)
server2, server2_vars = create_server(
server2_endpoint, server2_name, server2_namespace_uri, server2_node_dict
)

vup = VarUpdater(server1_vars, server2_vars)
```

References

1. Wang, S.; Wan, J.; Zhang, D.; Li, D.; Zhang, C. Towards smart factory for industry 4.0: A self-organized multi-agent system with big data based feedback and coordination. *Comput. Netw.* **2016**, *101*, 158–168. [CrossRef]
2. Wan, J.; Tang, S.; Shu, Z.; Li, D.; Wang, S.; Imran, M.; Vasilakos, A.V. Software-defined industrial internet of things in the context of industry 4.0. *IEEE Sens. J.* **2016**, *16*, 7373–7380. [CrossRef]
3. Cimino, A.; Gnoni, M.G.; Longo, F.; La Rosa, A. Digital Twin (DT) based methodology to support effective design of industrial production lines. *Procedia Comput. Sci.* **2023**, *217*, 1896–1907. [CrossRef]
4. Rasheed, R.; Palaiyah, S.; Maryna, C.; Iryna, K.; Olena, I.; Pant, B. Cyber-Physical protection system (CPPS) for improving key performance indicator (KPI) in a smart factory. In Proceedings of the AIP Conference Proceedings, Chennai, India, 2 August 2024; Volume 2937. [CrossRef]
5. Bai, Y. Industrial Internet of things over tactile Internet in the context of intelligent manufacturing. *Clust. Comput.* **2018**, *21*, 869–877. [CrossRef]
6. Goecks, L.S.; Habekost, A.F.; Coruzzolo, A.M.; Sellitto, M.A. Industry 4.0 and smart systems in manufacturing: Guidelines for the implementation of a smart statistical process control. *Appl. Syst. Innov.* **2024**, *7*, 24. [CrossRef]
7. Moshiri, A.; Hemmatyar, A.M.A. OPC UA over TSN (Time Sensitive Network) for Vertical and Machine to Machine Communication. In Proceedings of the 2023 9th International Conference on Control, Instrumentation and Automation (ICCIA), Tehran, Iran, 20–21 December 2023. [CrossRef]
8. Trifonov, H.; Heffernan, D. OPC UA TSN: A next-generation network for Industry 4.0 and IIoT. *Int. J. Pervasive Comput. Commun.* **2023**, *19*, 386–411. [CrossRef]

9. Ho, C.L.D.; Lung, C.H.; Mao, Z. Comparative Analysis of Real-Time Data Processing Architectures: Kafka versus MQTT Broker in IoT. In Proceedings of the 2024 IEEE 4th International Conference on Electronic Communications, Internet of Things and Big Data (ICEIB), Taipei, Taiwan, China, 19–21 April 2024. [CrossRef]
10. Takefusa, A.; Sun, J.; Fujiwara, I.; Yoshida, H.; Aida, K.; Pu, C. SINETStream: Enabling research iot applications with portability, security and performance requirements. In Proceedings of the 2021 IEEE 45th Annual Computers, Software, and Applications Conference (COMPSAC), Madrid, Spain, 12–16 July 2021. [CrossRef]
11. Sharanya, S.; Venkataraman, R.; Murali, G. Edge AI: From the perspective of predictive maintenance. In *Applied Edge AI*; Auerbach Publications: Boca Raton, FL, USA, 2022; pp. 171–192.
12. Han, S.I.; Lee, D.S.; Han, J.H.; Shin, H.J. The Design of Smart Factory System using AI Edge Device. *J. Korea Inst. Inf. Electron. Commun. Technol.* **2022**, *15*, 257–270. [CrossRef]
13. Li, Z.; Mei, X.; Sun, Z.; Xu, J.; Zhang, J.; Zhang, D.; Zhu, J. A reference framework for the digital twin smart factory based on cloud-fog-edge computing collaboration. *J. Intell. Manuf.* **2024**, *36*, 3625–3645. [CrossRef]
14. Singh, R.; Gill, S.S. Edge AI: A survey. *Internet Things Cyber-Phys. Syst.* **2023**, *3*, 71–92. [CrossRef]
15. Hu, L.; Miao, Y.; Wu, G.; Hassan, M.M.; Humar, I. iRobot-Factory: An intelligent robot factory based on cognitive manufacturing and edge computing. *Future Gener. Comput. Syst.* **2019**, *90*, 569–577. [CrossRef]
16. Roopaei, M.; Rad, P.; Choo, K.K.R. Cloud of things in smart agriculture: Intelligent irrigation monitoring by thermal imaging. *IEEE Cloud Comput.* **2017**, *4*, 10–15. [CrossRef]
17. Chen, W. Intelligent manufacturing production line data monitoring system for industrial internet of things. *Comput. Commun.* **2020**, *151*, 31–41. [CrossRef]
18. Leitão, P.; Rodrigues, N.; Barbosa, J.; Turrin, C.; Pagani, A. Intelligent products: The grace experience. *Control Eng. Pract.* **2015**, *42*, 95–105. [CrossRef]
19. Chen, B.; Wan, J.; Shu, L.; Li, P.; Mukherjee, M.; Yin, B. Smart factory of industry 4.0: Key technologies, application case, and challenges. *IEEE Access* **2017**, *6*, 6505–6519. [CrossRef]
20. Colombo, A.W.; Karnouskos, S.; Kaynak, O.; Shi, Y.; Yin, S. Industrial cyberphysical systems: A backbone of the fourth industrial revolution. *IEEE Ind. Electron. Mag.* **2017**, *11*, 6–16. [CrossRef]
21. Givehchi, O.; Landsdorf, K.; Simoens, P.; Colombo, A.W. Interoperability for industrial cyber-physical systems: An approach for legacy systems. *IEEE Trans. Ind. Inform.* **2017**, *13*, 3370–3378. [CrossRef]
22. Oliveira, L.E.S.D.; Álvares, A.J. Axiomatic Design Applied to the Development of a System for Monitoring and Teleoperation of a CNC Machine through the Internet. *Procedia Cirp* **2016**, *53*, 198–205. [CrossRef]
23. Botta, A.; De Donato, W.; Persico, V.; Pescapé, A. Integration of cloud computing and internet of things: A survey. *Future Gener. Comput. Syst.* **2016**, *56*, 684–700. [CrossRef]
24. Atri, P. Design and Implementation of High-Throughput Data Streams using Apache Kafka for Real-Time Data Pipelines. *Int. J. Sci. Res. IJSR* **2018**, *7*, 1988–1991. [CrossRef]