




A Distributed Real-Time Monitoring Scheme for Air Pressure Stream Data Based on Kafka

Zixiang Zhou ¹ , Lei Zhou ¹  and Zhiguo Chen ^{1,2,*} 

¹ School of Computer Science, School of Cyber Science and Engineering, Nanjing University of Information Science and Technology, Nanjing 210044, China; zhouzixiang@nuist.edu.cn (Z.Z.); zhoulei@nuist.edu.cn (L.Z.)

² Engineering Research Center of Digital Forensics, Ministry of Education, Nanjing University of Information Science and Technology, Nanjing 210044, China

* Correspondence: chenzhiguo@nuist.edu.cn

Abstract: Strict air pressure control is paramount in industries such as petroleum, chemicals, transportation, and mining to ensure production safety and to improve operational efficiency. In these fields, accurate real-time air pressure monitoring is critical to optimize operations and ensure facility and personnel safety. Although current Internet of Things air pressure monitoring systems enable users to make decisions based on objective data, existing approaches are limited by long response times, low efficiency, and inadequate preprocessing. Additionally, the exponential increase in data volumes creates the risk of server downtime. To address these challenges, this paper proposes a novel real-time air pressure monitoring scheme that uses Arduino microcontrollers in conjunction with GPRS network communication. It also uses Apache Kafka to construct a multi-server cluster for high-performance message processing. Furthermore, data are backed up by configuring multiple replications, which safeguards against data loss during server failures. The scheme also includes an intuitive and user-friendly visualization interface for data analysis and subsequent decision making. The experimental results demonstrate that this approach offers high throughput and timely responsiveness, providing a more reliable option for real-time gathering, analysis, and storage of massive data.

Keywords: IoT; real-time processing; distributed stream processing; Apache Kafka; monitoring



Citation: Zhou, Z.; Zhou, L.; Chen, Z. A Distributed Real-Time Monitoring Scheme for Air Pressure Stream Data Based on Kafka. *Appl. Sci.* **2024**, *14*, 4967. <https://doi.org/10.3390/app14124967>

Academic Editor: Eui-Nam Huh

Received: 16 April 2024

Revised: 28 May 2024

Accepted: 4 June 2024

Published: 7 June 2024



Copyright: © 2024 by the authors. Licensee MDPI, Basel, Switzerland. This article is an open access article distributed under the terms and conditions of the Creative Commons Attribution (CC BY) license (<https://creativecommons.org/licenses/by/4.0/>).

1. Introduction

With continuous technological advancement and increasing demand, real-time air pressure monitoring has become indispensable and crucial in various industries. Strict air pressure control is essential in sectors such as petroleum, chemicals, transportation, and mining [1–4]; an accurate, timely, and remotely controllable real-time pressure monitoring scheme is of paramount importance. This would enable management personnel to make rapid decisions in response to emergencies, directly impacting production safety and operational efficiency.

The recent widespread implementation of Industry 4.0 has initiated a new wave of digital industrial development. This paradigm can deeply impact efficiency, energy, sustainability, working conditions, human resources, production management, and maintenance planning in various industries [5]. Comparative analyses conducted by Domínguez-Bolaño et al. [6] on widely used open-source Internet of Things (IoT) platforms, as well as the intelligent factory architecture based on Industry 4.0 technologies implemented through open-source software by Fortoul-Diaz et al. [7], demonstrate that the various free and open-source IoT platforms and projects can provide reliable technical support for the development of Industry 4.0. These platforms offer advantages to include low cost, internet-accessible information, and robust configuration capabilities for developers.

Simultaneously, the development of IoT technology has allowed various types of low-power sensor devices to incorporate innovative technologies such as electronic minia-

turization, portability, wireless communication, and energy-efficient microprocessors [8,9]. Waworundeng et al. [10] proposed a tire pressure detection system for motorized vehicles. Fay et al. [11] developed a landfill gas pressure monitoring and management system. Hassan et al. [12] devised an environmental monitoring system. These systems use low-power sensor networks to wirelessly transmit real-time air pressure data to provide remote, real-time, and reliable data transmission.

However, traditional air pressure monitoring systems have some limitations. Typically, they store the real-time air pressure data gathered by sensors in databases. Subsequent analyses and decision making depend on reading, querying, and processing the stored data from the database. The system faces some challenges. Firstly, user feedback depends on database operations, which introduces delay. Data volume increases cause performance bottlenecks, which further exacerbate this delay. This implies that users cannot promptly access real-time air pressure data during emergencies, causing difficulties when making corresponding decisions quickly, which in turn increases the possibility of data loss. Therefore, the importance of real-time monitoring systems cannot be overlooked in practical working environments; the systems must be capable of quickly gathering and analyzing large amounts of data streams generated by sensors. This process becomes crucial when dealing with time series data to support critical decisions [13]. However, overcoming the limitations of existing schemes is a significant challenge in real-time air pressure monitoring. Therefore, we must continuously explore new methods and technologies to address these challenges, improve system performance and reliability, and better meet the specific requirements of real-time monitoring systems for data gathering, processing, and supporting decisions.

The emergence of complex event processing (CEP) and distributed stream processing (DSP) systems has provided a new approach to processing high-volume stream data using big data technology in real time [13]. In this field, Kafka plays an important role as a stream processing (SP) architecture based on the pub/sub mechanism, especially for handling massive logs in distributed environments with multiple sensor nodes and large clusters [14]. Therefore, Kafka has become widely used as a stream processing tool for real-time data analysis [15]. Kafka has several advantages, including real-time processing, strong scalability, low delay, and high flexibility [16].

This paper proposes a distributed real-time monitoring scheme for air pressure stream data based on Kafka. This scheme can customize Kafka clusters according to the number of sensor nodes in the working environment and can transmit real-time air pressure data to Kafka clusters via producers. These clusters are then read and processed by consumers and fed back to the client. The data are stored in a database for historical-record-based decision analyses. Through this scheme, users can obtain high throughput, low delay, and reliable real-time air pressure monitoring services.

The main contributions of this paper are as follows:

- We integrate the advantages of Apache Kafka and the characteristics of distributed processing, leading to the proposal of a distributed real-time SP scheme for air pressure monitoring that is tailored to handle massive data. This integration allows the system to efficiently handle large-scale stream data and provide reliable real-time monitoring. The reliability and efficiency of this scheme in multi-sensor working environments are validated through performance comparisons with communication via ActiveMQ pipelines and existing systems, further demonstrating its practical feasibility.
- We construct a low-power sensor network using low-cost microcontrollers, pressure sensors, and communication modules to establish a reliable hardware foundation for the practical application of the scheme.
- We provide a visualization interface and interactive data feedback platform through web pages and apps, allowing users to intuitively understand pressure monitoring data and perform real-time data interaction and analysis. The processed data are persistently stored in the MongoDB database, which enables users to conduct decision analyses based on historical data. This provides users with deeper and more

comprehensive data analysis capabilities to better support decision making and the application of real-time monitoring systems.

This paper is organized as follows: Section 2 discusses the literature, providing the background and an overview of the field; Section 3 introduces the overall architecture of the scheme, including the hardware for the sensor devices and system architecture and the functionalities of each component; Section 4 tests the throughput and transmission delay of Kafka under a large-scale data background and makes comparisons to ActiveMQ message brokers; finally, Section 5 summarizes the paper, provides conclusions, and suggests potential future research directions.

2. Related Works

With the advancement of IoT technology and cyber-physical systems, human society is gradually entering the Trillion Sensors (TSensors) era [17]. In fields such as real-time environmental monitoring that involve digital sensors and IoT, the value will sharply decline if stream data cannot be effectively processed. Therefore, real-time monitoring systems must be able to process continuous data streams without extreme delays to ensure the timeliness of data [18]. The challenge of developing a rapid and efficient large-scale stream-data processing method is a critical issue in this field.

Traditional centralized computing systems consolidate processors, memory, storage, and other resources at a central node, which requires expensive hardware to support large-scale data processing and multi-user usage [19]. In contrast, distributed computing systems (DCSs) leverage multiple resources distributed across a network: using inexpensive standard hardware to process vast amounts of data and complex operations. DCS offers advantages in information sharing, concurrent processing, and computational capabilities over centralized computing systems [20]. The importance of DSP systems has correspondingly increased as modern applications impose stricter time constraints on the real-time analysis of massive data gathered by sensors [21,22].

Compared to traditional static data batch processing, SP is a dynamic process that continuously integrates new data to compute results. SP primarily analyzes data streams from event producers such as sensors, devices, automation stations, and relational database systems through data analysis platforms built on relevant engines and infrastructure to detect and extract meaningful patterns and events [13]. The comparative study by Isah et al. [23] on distributed data SP and analysis frameworks, the proposal by Pajarola et al. [24] of a stream-based point processing framework, and the performance measurements of distributed stream data processing systems conducted by Karimov et al. [25] all indicate that distributed stream data processing systems possess advantages to include low delay, high performance, high throughput, and scalability when handling large amounts of real-time data. Additionally, the smart grid monitoring architecture based on DSP proposed by Carvalho et al. [26] demonstrates that DSP can be applied in real-time monitoring. Therefore, DSP can effectively handle real-time data monitoring.

This paper combines DSP with a real-time air pressure monitoring system to improve real-time environmental monitoring. At present, SP tools like Apache Flink, Apache Kafka, Apache Storm, and ActiveMQ play an important role in ingesting, processing, storing, indexing, and managing stream data. Apache Kafka and ActiveMQ are based on pub/sub structures and effectively solve the blocking problem caused by numerous asynchronous messages. Therefore, this paper designs real-time air pressure stream data monitoring schemes based on Apache Kafka and ActiveMQ and analyzes and compares their throughput and transmission delay.

Apache Kafka is a distributed-event SP engine that consists of multiple components, including records or messages, record streams, consumers, producers, brokers, logs, partitions, and clusters [23]. In Kafka, producers send real-time data streams to different partitions of the corresponding topics in the Kafka cluster, while consumers fetch data from these topics and associate them with predefined thresholds for predictive analyses. Kafka uses a persistent querying system (KSQL) to provide pipeline-style real-time data stream

processing, eliminating the need to directly submit data streams to the database [27] or to read, query, and process stored data from the database, thereby ensuring the timeliness of data transmission. Kafka provides a low-delay, high throughput, and fault-tolerant publish–subscribe pipeline based on pub/sub structures, making it suitable for processing massive data in a distributed environment. Therefore, it can provide a high-performance information channel for real-time air pressure monitoring.

Kafka producers send data to the corresponding topics in the Kafka cluster, while consumers read data from their subscribed topics (Figure 1). A Kafka cluster consists of several brokers; all partitions of the same topic are distributed across different brokers. For each partition, one replica is elected as the leader, and the others are followers. The leader of each partition maintains an in-sync replica (ISR) list to ensure data consistency. When a producer sends a message, the message is first written to the leader partition and is then replicated to all replicas in the ISR. Kafka uses this replication mechanism to achieve fault-tolerant message transmission, ensuring the cluster’s ability to recover data in case of a broker failure [28]. ZooKeeper serves as the manager responsible for managing the cluster state and metadata and coordinating partition assignment and fault recovery.

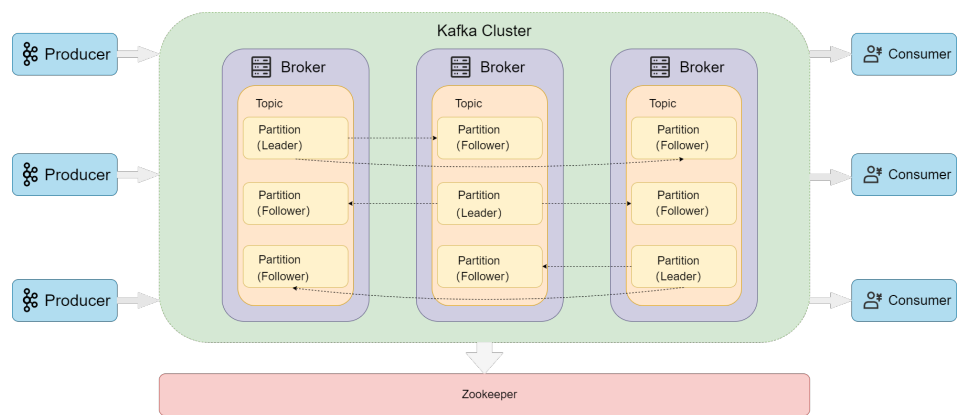


Figure 1. Kafka message-passing principle.

The ActiveMQ message broker consists of connections, producers, consumers, messages, and message queues. The broker enables data transmission between producers and consumers through message queuing services. In ActiveMQ, producers send real-time data to message queues, which consumers then retrieve. If there is only one consumer in the working environment, the destination of data transmission will be a queue; if there are multiple consumers, messages are routed using the pub/sub model [29]. In this process, consumers send an acknowledgment (ACK) message to the producer through the ACK mechanism, which indicates that the transmitted data have been received and verified without errors, ensuring message reliability. ActiveMQ also supports message transformation, which converts messages from one format to another during routing or consumption. This enables seamless integration between systems with different message formats or protocols [30]. Figure 2 illustrates the message-passing principle of ActiveMQ.

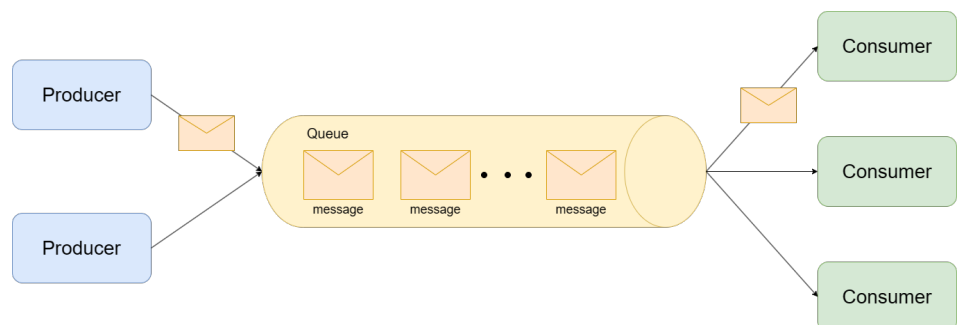


Figure 2. ActiveMQ message-passing principle.

3. Design of Real-Time Monitoring Method

3.1. Hardware

We constructed a low-power sensor network to efficiently and reliably transmit real-time air pressure data. The hardware devices used are shown in Figure 3.

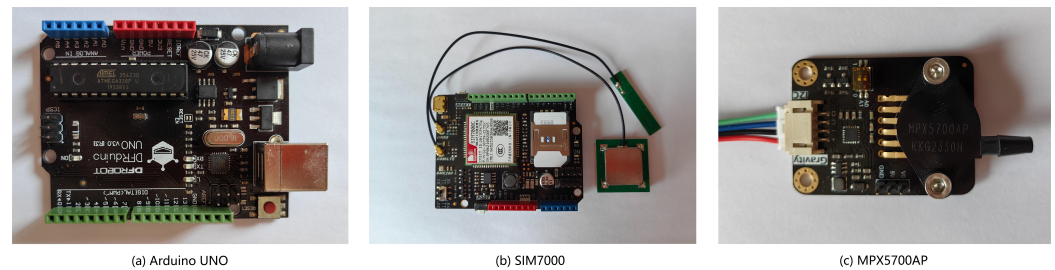


Figure 3. Hardware components used in the sensor network.

The Arduino Uno is based on the ATmega328 microcontroller and features 14 digital input/output pins, 6 analog inputs, and a USB connection interface; the hardware specifications are outlined in Table 1. We can execute the code by directly uploading sensor code to the EEPROM of the Arduino Uno without the need for any external software connection to a computer, which ensures that the device can independently gather real-time data and maintain high portability [31]. We constructed a low-cost, low-power, and scalable real-time monitoring platform tailored for air pressure data streams by integrating the communication functionality of wireless communication modules and the data acquisition capabilities of pressure sensor modules into the Arduino Uno microcontroller.

Table 1. Specifications of Arduino Uno.

Content	Specification
Microcontroller	ATmega328
Working voltage	5 V
CPU frequency	16 MHz
EEPROM	1 KB
Flash	32 KB
SRAM	2 KB
Dimensions	75 × 55 × 15 mm

The SIM7000 wireless communication module supports multi-band NB-IoT, dual-band GPRS/EDGE, and LTE-FDD communication. Sensor devices need to gather and transmit data frequently to monitor in real time. Therefore, we have opted to use the GPRS network and transmit real-time data to Kafka via UDP communication to provide an efficient data transmission channel that facilitates prompt responses to the data sent by sensor devices.

The MPX5700AP sensor is a single-port silicon pressure sensor integrated into a six-pin SIP package [10]. It supports I2C digital output, allowing calibration based on known air pressure values, and can quickly and accurately measure pressure values in various environments, such as pipelines. Its parameters are listed in Table 2. In this scheme, the microcontroller and pressure sensor are connected via the I2C bus communication protocol to gather real-time air pressure data.

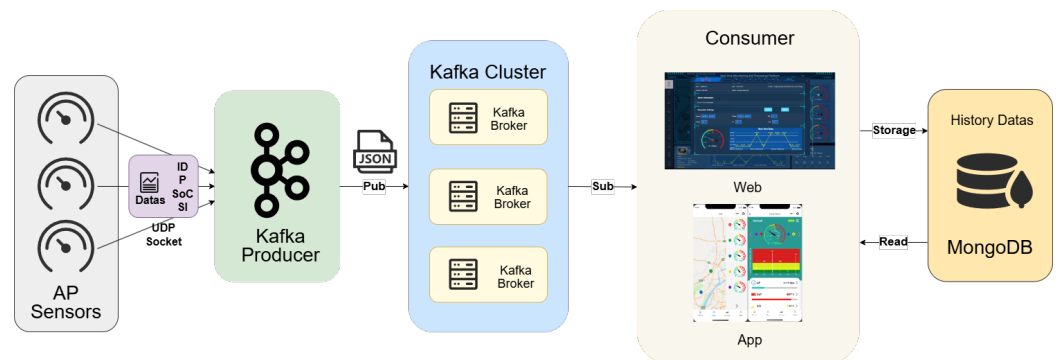
Table 2. Specifications of MPX5700AP.

Content	Specification
Working voltage	$3.3 \leq 5.5$ V DC
Power dissipation	0.06 W (5 V)
Output signal	I2C output ($0 \leq 3$ V)
Measurement range	15–700 kPa
Resolution ratio	≤ 1 kPa

3.2. Framework Design

With the advent of the Industrial IoT (IIoT) and industrial analytics, many new application scenarios have emerged for which critical decisions rely on the large-scale analysis of sensor streams [32]. As data volumes grow exponentially, applications must handle larger amounts of unstructured and complex datasets simultaneously. Relational database models (such as MySQL v5.7.44) have severe limitations when dealing with such massive data [33]. Non-relational databases, conversely, are well-suited for large-scale data processing due to their flexible structures, high performance, fault tolerance, and adaptability to dynamic schema. Among these, MongoDB, based on dynamic schema JavaScript Object Notation (JSON) documents, outperforms MySQL in insertions, queries, updates, and deletions [34], making it more suitable for the real-time monitoring method proposed in this paper. Therefore, we chose MongoDB v4.4.6 to store the processed data.

In real-time data streaming scenarios, a smaller number of brokers can significantly reduce end-to-end delay due to the reduced complexity of cluster management processes [35]. For data packets of the same size, more partitions lead to an incremental increase in transmission intervals. This is because Kafka producers need more time to arrange the metadata for each partition [28]. Therefore, we configured the Kafka cluster with three brokers and divided the message transmission topic into three partitions. The overall framework of the proposed real-time pressure monitoring scheme consists of a sensor network module, a producer module, a cluster module, consumer modules, and a database module (Figure 4).

**Figure 4.** Real-time air pressure monitoring method framework.

The sensor network module consists of multiple sensor devices distributed in the working environment. Each device consists of an Arduino Uno microcontroller combined with an MPX5700AP pressure sensor that gathers real-time environmental data. Using the SIM7000 communication module, the sensor sends real-time data, including the sensor ID, pressure value, battery level, and signal strength, in string format via the UDP protocol to the producer.

Then, the producer module splits and integrates the data strings from the sensor network module into JSON format and sends the data to the specified topic in the Kafka cluster according to the arrival time.

Next, the cluster module is distributed on a Kafka cluster consisting of three servers: each acting as a Kafka broker. The message topics are distributed among the three Kafka brokers and are divided into three partitions. For each broker, one partition is designated as

the leader, while the remaining partitions are followers. The JSON data from the producer module are written to the leader partition and then replicated to the follower partitions.

Then, the consumer modules receive the JSON data from the cluster module based on the subscribed topics and perform predictive analyses on them according to predefined thresholds. We designed a web and app interface (Figures 5 and 6) to provide users with visualization interfaces for sensor management, abnormal data alerting, historical data querying, and real-time stream data monitoring.

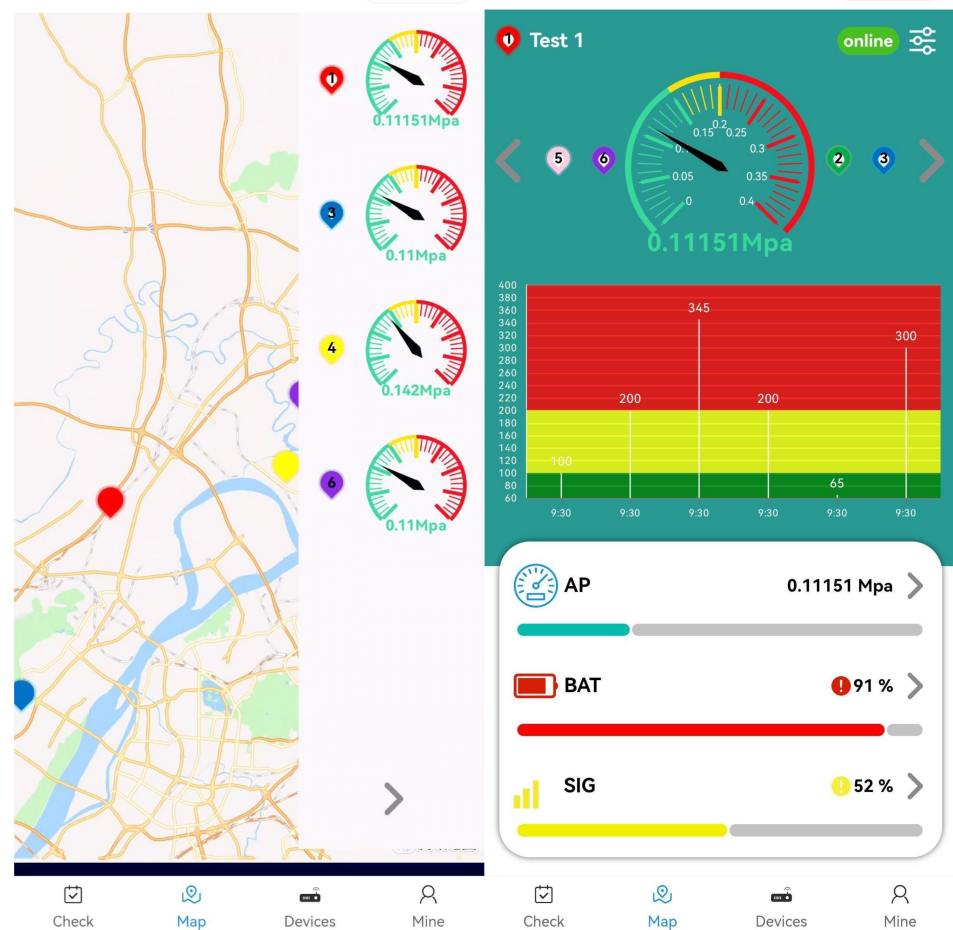


Figure 5. App data feedback.

Finally, the database module runs on another server and receives data from consumer modules; these data are persistently stored according to predefined storage policies. The consumer modules can retrieve historical data from the database module to formulate and modify alert and alarm strategies.

The framework is designed to host data from sensors connected to equipment other than Arduino. For example, if a pressure sensor is connected to a data acquisition unit, the sensor ID serves as a unique identifier for different devices connected to the producer. This enhances the industrial applicability of the framework.

Message blocking may lead to transmission delays in practical work environments where many sensors are simultaneously connected to one producer. Therefore, users can customize the number of producers based on the scale of the sensor network. Additionally, the number of consumers in the framework depends on the concurrent user count; when a user requests data, the system will initiate a consumer process to retrieve the data.

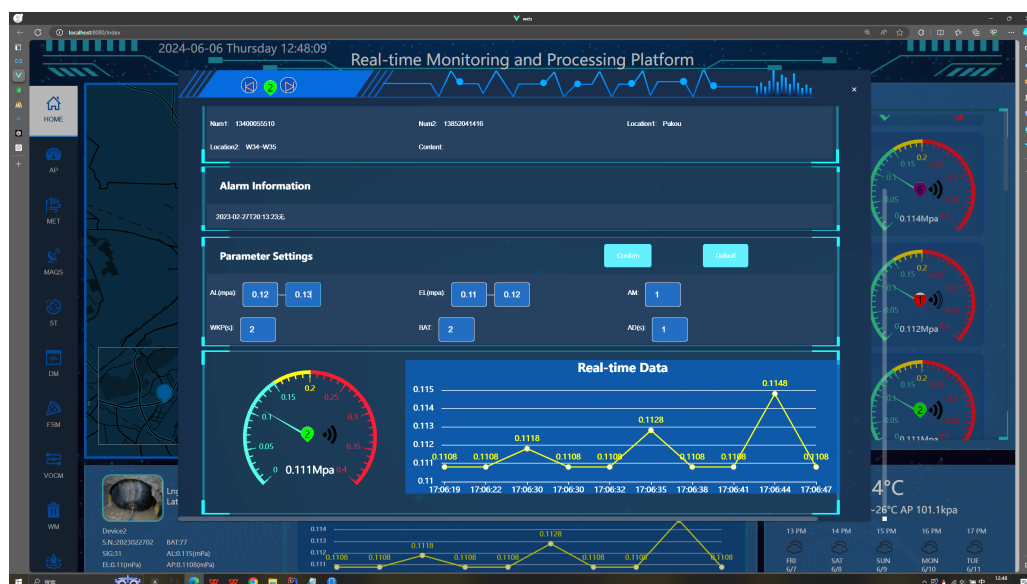


Figure 6. Web page data feedback.

4. Experimental Results and Analysis

The scheme requires environmental air pressure values to be transmitted to the consumers using a high-speed message processing framework to meet the real-time processing requirements of massive data. Kafka and ActiveMQ are two widely used message queuing systems [36]. Therefore, to compare the suitabilities of different SP tools in the scheme, we compare the throughput and transmission delays between Kafka and ActiveMQ to determine the system's capacity and end-to-end message transmission time. This analysis allows us to evaluate the real-time processing performance for the data and helps us to understand the performance of the system under different message transmission frameworks in order to select a messaging framework suitable for the scheme.

4.1. System Environment

We used the computer hardware and software configurations listed in Table 3 to gather sensor stream data in practical work environments with the aim of validating the performance achieved by Kafka and ActiveMQ in the real-time air pressure stream data monitoring scheme.

Table 3. Hardware and software environment.

Content	Specification
CPU	Intel® Core™ i9-9900K 3.6 GHz
Memory	32 GB
SSD	500 GB
OS	2.12-2.4.1
Apache Kafka	Windows 10 Version 22H2 (OS build 19045)
Apache ZooKeeper	3.5.7
Java	1.8.0_321
ActiveMQ	5.16.3

4.2. Transfer Data

To monitor air pressure, sensors continuously gather real-time air pressure data and transmit it to producers via the UDP communication protocol. The producers are responsible for converting these data into a structured message, which includes key information including the sensor ID, pressure value, battery level, and transmission signal strength. Once the producer creates the message, it is sent to the cluster or message queue. Consumers can subscribe to the transmission topics as needed to receive the required pressure

data. The data structures transmitted by Kafka and ActiveMQ for the performance comparison are in an easily parsed and processable JSON format (Table 4). Additionally, to consider transmission efficiency and data integrity, the size of each message is limited to 40 bytes.

Table 4. Transfer data.

“ID”: “001”, “P”: 101.95, “SoC”: 77, “SI”: 30

4.3. Throughput Comparison between Kafka and ActiveMQ

We conducted an analysis and comparison of the performance of Kafka and ActiveMQ from the perspective of data throughput. We ran performance testing scripts for producers and consumers simultaneously on the same computer node to evaluate their relative throughput performance in the same environment. This approach simulated the scenario wherein producers and consumers work simultaneously in the practical system, which provides a better reflection of the overall performance of the scheme.

In throughput testing, the number of messages transmitted per second (msg/s) is a key metric to measure the transmission rate. To evaluate the effective throughput of the messaging middleware, we adopted a standard method: measure the number of messages sent at 5 s intervals while the producer sends messages, which thus calculates the average message-sending rate per second. The throughput of consumers is also measured in the same manner. This method provides a more comprehensive and accurate assessment because it considers the average rate of message transmission rather than relying solely on instantaneous performance. Comparing the throughput of Kafka and ActiveMQ producers and consumers in this manner will help us better understand their performance differences in practice.

Kafka provides developers with two performance testing scripts to evaluate the performance of producers and consumers in the system. The performance test script for producers aims to measure the message throughput transmitted per second through partitions; the performance test script for consumers measures the message throughput received per second from partitions. The ActiveMQ installation package does not contain performance test scripts like Kafka; therefore, we created producers and consumers by building a Maven project to measure the relative throughput of ActiveMQ.

The relative throughput test results of Kafka and ActiveMQ (Figure 7) demonstrate that Kafka has a significant advantage in throughput. Specifically, Kafka’s producer average throughput reached 3,096,134 msg/s, which far surpassed ActiveMQ’s 71,902 msg/s. Similarly, Kafka’s consumer average throughput was 3,090,558 msg/s, which is significantly higher than ActiveMQ’s 72,500 msg/s. This comparison highlights Kafka’s efficiency and speed in message processing.

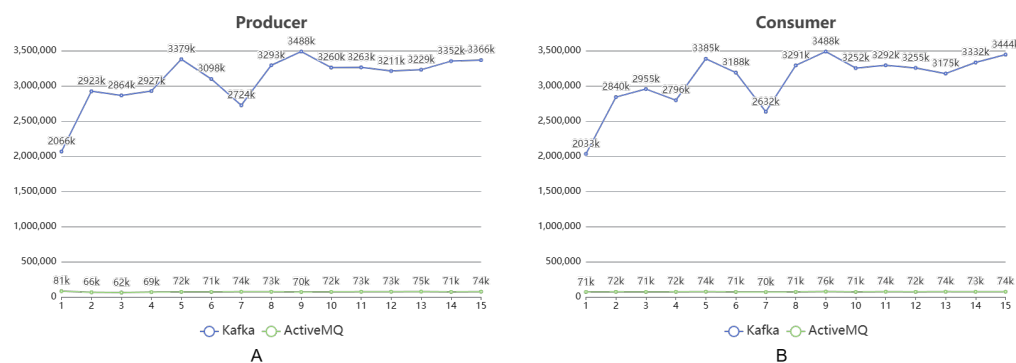


Figure 7. Throughput comparison (Producer and Consumer).

However, we also observed a certain degree of fluctuation in Kafka’s performance data. The standard deviations for Kafka’s producer and consumer were 359,370 and 385,038, respectively. Conversely, ActiveMQ exhibited a more stable throughput, with standard

deviations of 4389 for the producer and 1695 for the consumer. Despite Kafka's superiority in throughput over ActiveMQ, ActiveMQ is more reliably stable. This can be explained by the following factors. Firstly, Kafka employs a batch-based transmission method that transmits and consumes multiple messages at once, whereas ActiveMQ uses a pipelined message queue approach to pull and consume individual messages sequentially. Secondly, considering scenarios where there may be numerous messages but consumers cannot process them, ActiveMQ provides a quality of service (QoS) limiting mechanism that effectively controls message transmission efficiency and avoids message blocking. For example, in test A, the average message transmission rate of the producer in the first stage was 81,431 msg/s, but it dropped to 66,010 msg/s in the next stage, which demonstrates that the QoS limiting mechanism reduced the producer's message transmission rate. Lastly, ActiveMQ ensures message delivery stability by acknowledging when a message is sent.

Compared to Kafka, the performance of ActiveMQ's producer may be limited when dealing with large-scale, high-speed data streams owing to the QoS mechanism and message confirmation method during message transmission. Conversely, Kafka has a scalable storage layer and higher transmission and storage efficiency, can adapt to different scales and load requirements, and maintains high performance even when dealing with numerous data sources [37]. Therefore, considering Kafka's throughput, performance, and scalability and its ability to adapt to different application scenarios, Kafka can bring higher performance, better scalability, and more reliable message delivery capabilities to the system as the messaging middleware.

4.4. Delay Comparison between Kafka and ActiveMQ

We conducted tests and comparisons between Kafka and ActiveMQ to compare the transmission delay by using different messaging middleware in the scheme. We aimed to understand the time taken for information to be transmitted under different numbers of devices, thereby determining a reasonable scale for the sensor network and evaluating the performance of different messaging middleware in the scheme.

We employed a combination of practical and simulated data to simulate the practical working environment in order to test the delay, with the aim of assessing the time required for different numbers of devices to send a message from the producer end to the consumer end via Kafka and ActiveMQ. This experiment consisted of six test groups, covering scenarios where 1000, 3000, 10,000, 30,000, 100,000, and 300,000 devices each transmitted a single message, which ensured the comprehensiveness and reliability of the test results.

In a single transmission, the sensor devices send data containing the device ID, pressure value, battery level, and signal strength to the producer. The producer consolidates these data into a JSON format message of 40 bytes, which is ultimately consumed by the consumer. We calculated the delay of information transmission by computing the time difference between the sending and receiving of the data.

The specific testing process is as follows. Firstly, we created producers and consumers for Kafka and recorded the time differences for each test group from when the producer sent the data to when the consumer received it. When the producer received real-time data from the sensors, the scheme integrated the real-time data with the simulated data into a data queue ready for transmission. Before sending the message, we recorded the start time. When the consumer pulled messages from the topic, the received data were processed; these data included information such as the topic, partition, offset, and data, which are encapsulated in ConsumerRecords [38]. In Kafka versions 0.9 and later, the consumer record includes a timestamp field [39]. Therefore, we defined the timestamp of the last message received by the consumer as the end time. We obtained the transmission delay by calculating the difference between the end and start times. The calculation method for ActiveMQ transmission delay is comparable to this process.

In the transmission delay test of Kafka and ActiveMQ, we conducted 10 tests for each test group and took the average of these tests as the final test result.

Figure 8 displays the delay comparison results between Kafka and ActiveMQ. For the six different test groups, the average delays for Kafka were 15.6 ms, 23.1 ms, 48.6 ms, 96.3 ms, 206.1 ms, and 336.1 ms, respectively; ActiveMQ’s average delays were 48.9 ms, 92.5 ms, 271.9 ms, 562.8 ms, 1557.6 ms, and 4374.5 ms, respectively. ActiveMQ’s delay was between 3 to 10 times that of Kafka. When the number of devices is small, the difference in delay between the two is small. However, as the number of devices increases, the difference gradually increases.

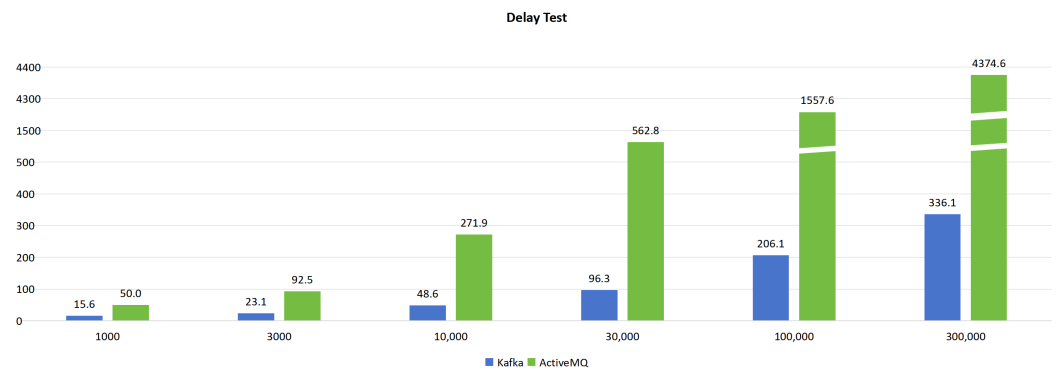


Figure 8. Delay comparison.

Batch-based Kafka offers lower transmission delays compared to ActiveMQ, which is based on a message queue. Kafka can handle more simultaneous data, resulting in lower propagation delays when processing the same amount of data. This makes Kafka more suitable for real-time processing of air pressure data.

4.5. Kafka Throughput Results and Analysis

Kafka demonstrates high throughput and low transmission delay in the real-time transmission of air pressure data streams. Therefore, we have selected Kafka as the messaging middleware for the distributed real-time air pressure data stream monitoring scheme. We conducted tests on the absolute throughput and relative throughput of the data streams to comprehensively evaluate the performance of the middleware. The aim was to identify the bottlenecks in system performance and improve the overall performance and reliability of the system through scheme enhancements.

Producers and consumers input and output messages through specified topics. In the topic settings, the partition specifies the practical storage method of the message. To effectively use a topic, multiple partitions must be defined. In Kafka’s pub/sub structure, the consumer determines the topic of the received message, and the number of consumers determines the number of messages received. As part of a general distributed system, Kafka provides system fault tolerance through replication settings. When the leader of a partition fails, Kafka can automatically select another leader for compensation to ensure the repeated reception or output of data. Therefore, the number of topic partitions and replication factors in Kafka’s broker have a significant impact on system performance. We determined the specific parameter configuration by setting four different topics. The topics created are listed in Table 5.

Table 5. Topic list in Kafka.

Topic ID	Configuration
1	3 partitions, 3 replications
2	1 partition, 3 replications
3	3 partitions, 1 replication
4	1 partition, 1 replication

In our performance analysis experiments, we divided the throughput tests for producers and consumers into two modes: independent and synchronous. In the independent test, we conducted separate performance tests for producers and consumers to ensure that they did not interfere with each other. This allowed us to accurately verify the performance of producers and consumers and provided a reference for the design of the scheme parameters. In the synchronous test, we simultaneously ran performance testing scripts for producers and consumers to evaluate their relative throughput performance across multiple nodes. This approach simulated the scenario wherein producers and consumers work simultaneously in the practical system, providing a better reflection of the overall performance of the scheme.

The results of independent and synchronous tests conducted on Topics 1–4 are depicted in Figure 9. The x-axis represents 5 s time intervals, while the y-axis represents the average number of messages over consecutive 5 s periods; for instance, in test A, the value corresponding to marker point 1 on the x-axis is 2,070,549. This indicates that during the initial 5 s of the test, the average rate at which the producer transmitted data to the Kafka broker through Topic 1 was 2,070,549 msg/s. The remaining points in the graph reflect the performances of the producer and consumer during specific time intervals.

The independent test results of A, C, E, and G demonstrate that the consumer’s throughput exceeded that of the producer. This phenomenon can be attributed to the Kafka controller’s intervention when the producer inputs messages into the partition, which subsequently impacts the production speed. Specifically, the controller must perform multiple operations. First, it manages partitions and replicas and sends related information to ZooKeeper. Second, for topics that contain multiple partition groups, the controller distributes messages to each partition. Additionally, the controller must also manage the replication mechanism to ensure that the message replicas of each partition are distributed to the various brokers according to the specified replication quantity. Simultaneously, the producer must spend a certain amount of time waiting for the confirmation message from the leader of each partition. Therefore, when receiving producer messages, Kafka has a higher delay owing to the operations of the controller and the time required for confirmation, thereby reducing the message transmission volume of the producer.

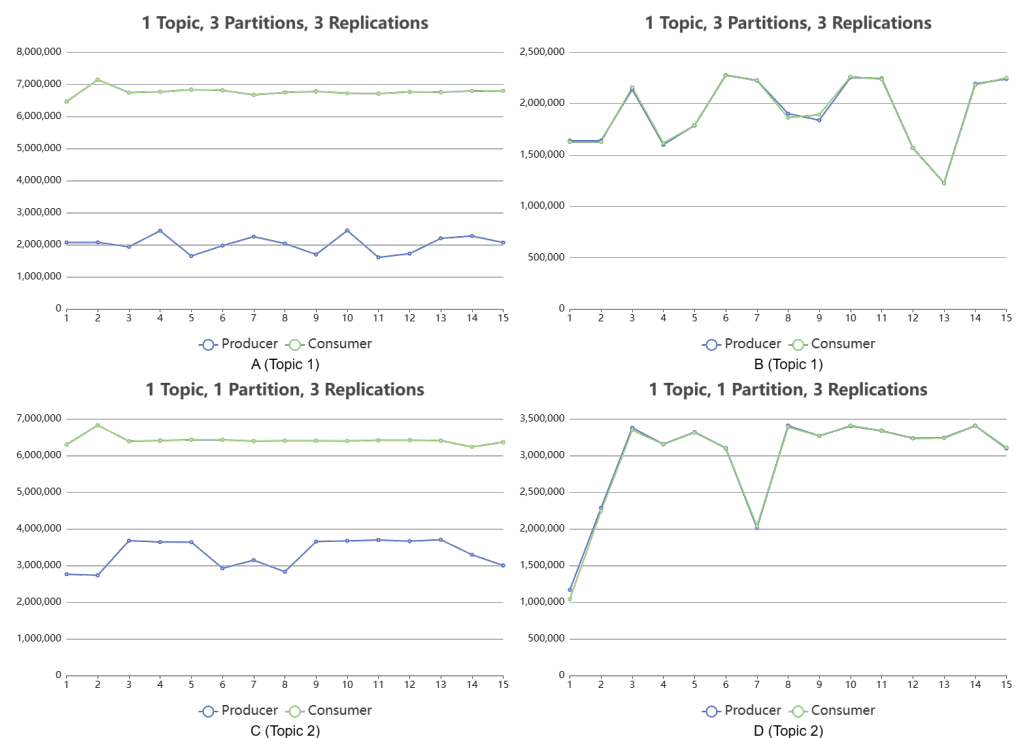


Figure 9. Cont.

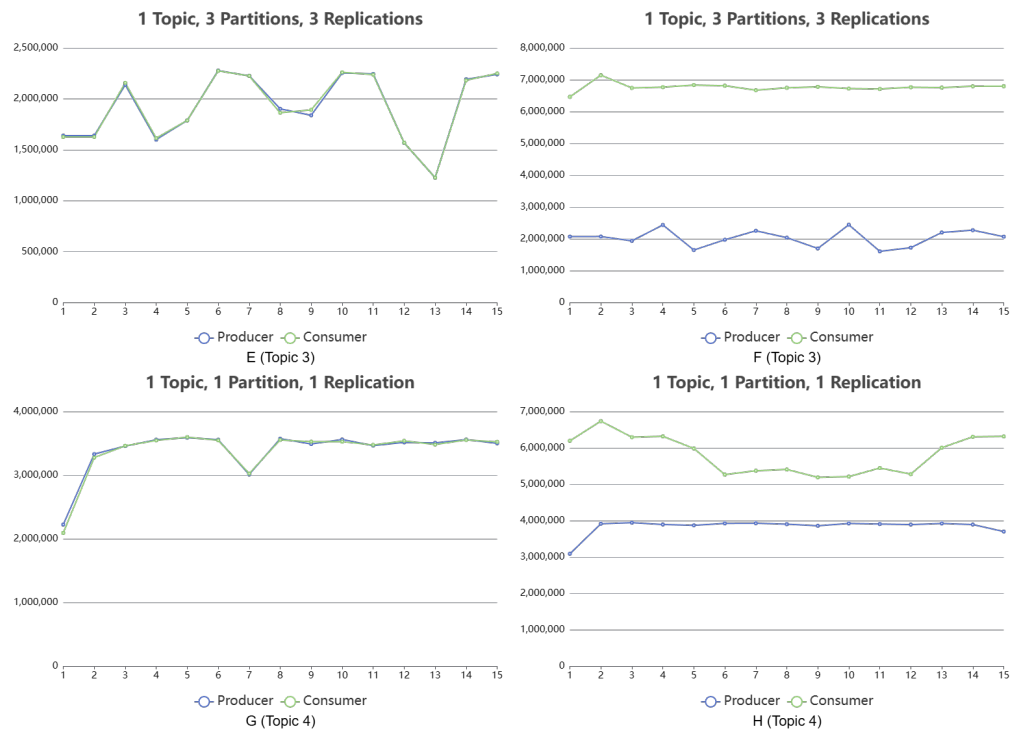


Figure 9. Throughput of Kafka.

Conversely, the consumers can request messages directly from ZooKeeper and retrieve messages assigned to specific topics using partition locations and offset information. Unlike producers, consumers can directly receive messages from the message storage location using sockets without the need for intermediary routing through the controller. This direct message retrieval process enables consumers to handle messages more efficiently and reduces processing delays, consequently exhibiting higher throughput compared to the producers.

The results of tests B, D, F, and H report that the relative throughput of the producers and the consumers is comparable. This can be explained by the mechanism of synchronous testing; in synchronous testing, the producer sends the message to the Kafka cluster, and then, the consumer can consume the message. Therefore, the consumer must wait for the producer’s message input, which results in the throughput of the producer and consumer being close. In other words, in synchronous testing, the throughput of the producer directly affects the throughput of the consumer. Additionally, compared to the test results in A, C, E, and G, the performance of the producer decreased because the producer needed to perform complex controller operations, replication operations, and ZooKeeper operations in synchronous testing; the performance of the producer decreased compared to independent testing.

The throughput of Kafka varied significantly depending on the configuration of different topics. This difference was primarily caused by the delay introduced by producers inputting messages. In the synchronous tests of the four topics listed in Table 5, Topic 4 (Figure H), with a single partition and single replication, performed the best. Topics 1 (Figure B) and 3 (Figure F) exhibited greater performance fluctuations and lower throughput compared to Topics 2 (Figure D) and 4 (Figure H), but they maintained overall high throughput. This phenomenon is attributed to the higher load on the controller when distributing messages across multiple partitions for Topics 1 and 3, leading to fluctuations in producer performance. Topic 1 (Figure B) performed worse than Topic 3 (Figure F), and Topic 2 (Figure D) performed worse than Topic 4 (Figure H): mainly due to the controller’s resource consumption during message replication, thereby reducing the overall performance of Topics 1 and 2. In the independent tests, the consumer throughput for Topics 1 (Figure A) and 3 (Figure E) was higher than that of Topics 2 (Figure C) and 4 (Figure G)

because the consumers in Topics 1 and 3 concurrently read messages from multiple partitions (3 partitions), thereby improving the overall message transmission performance.

4.6. Comparison with Existing Work

As shown in Table 6, to explore the performance of Kafka in the field of real-time monitoring, we compared our scheme with other real-time monitoring systems. By evaluating the technical approaches and performance data, we examined Kafka's performance and innovations in the real-time monitoring of air pressure.

Table 6. Performance comparison of different systems.

Authors	Middleware	Throughput (msg/s)	Delay
Buzzoni et al. [40]	ActiveMQ	-	15 ms (1 msg)
Happ et al. [41]	RabbitMQ	30k	1 ms (1 msg)
Berlian et al. [42]	Hadoop	-	85 ms (1k msg)
Buddhika et al. [43]	Granules	2000k	-
Han et al. [44]	Kafka	200k	-
Corral-Plaza et al. [45]	Kafka	200k	2.5 ms (1 msg)
Proposed work	Kafka	3000k	15.6 (1k msg)

Buxton et al. [40] implemented an IoT-based airfield lightning system using an AMQP pub/sub broker. This system uses ActiveMQ to receive events from the airport lighting control and monitoring system to support monitoring functions and subscribes to ActiveMQ to receive commands from third-party clients. Tests showed that the system's delay in transmitting a single message is 15 ms. Similarly, Happ et al. [41] studied the maximum sustainable throughput and delay of different pub/sub systems under realistic load conditions using traces from real sensors. The results indicated that for weather monitoring scenarios, RabbitMQ achieves a maximum throughput of 30k msg/s, with approximately 70% of the measured delay values being below 1 ms. These studies utilized AMQP based on the queue model, which may face performance limitations when handling large volumes of high-speed data streams. In contrast, Kafka, as a distributed message queue system, captures data streams from event sources in real time, enabling our scheme to achieve higher throughput and lower delays.

Berlian et al. [42] designed a real-time intelligent environmental monitoring and analysis system framework based on underwater IoT and big data. This framework stores and analyzes data gathered from remotely operated vehicles and water parameter sensors on a multi-node Hadoop-cluster data-center platform. By utilizing Hive to handle larger data volumes, the framework can query one thousand records within 85 ms. Buddhika et al. [43] developed the NEPTUNE system for real-time, high-throughput stream processing in IoT and sensor environments. This system is built on the Granules computing framework [46] and uses a pull-based method to ingest streams from IoT gateways. The stream processors then process the data and emit stream packets through one or more outgoing streams, achieving a throughput of 2000k msg/s with 93.7% bandwidth utilization. These systems transmit data through multiple steps, leading to a relatively complex processing flow. In comparison, our approach simplifies message processing by directly sending real-time data to Kafka producers, enabling faster data feedback.

Han et al. [44] implemented the RT-DAP platform on Microsoft Azure for real-time data analysis in large-scale industrial process monitoring and control. This platform combines Kafka and Storm to process time series data in real time and feeds the results back to the physical system or into a database. Tests showed that the throughput of the platform was around 200,000 msg/s when the payload size was smaller than 64 bytes. In contrast, Corral-Plaza et al. [45] proposed an architecture for handling heterogeneous data sources in the IoT. By integrating multiple data sources using Kafka and employing a CEP engine to extract relevant events, they achieved a processing speed of 200k msg/s for data sizes of 320 bytes per record, with an average transmission delay of 25 ms per message.

In comparison, our scheme formats the transmitted data into JSON and limits the size to 40 bytes, significantly enhancing system performance.

Therefore, the real-time air pressure monitoring scheme proposed in this paper utilizes a distributed Kafka stream processing engine to transmit real-time sensor data. The delay for transmitting one thousand messages is as low as 15.6 ms, and the throughput reaches up to 3000k msg/s. These results demonstrate that our scheme offers significant advantages in terms of transmission delay and throughput and exhibits excellent performance and potential application value in the field of real-time air pressure monitoring with large data volumes. This can provide a valuable reference and insights for related research and applications.

5. Conclusions

This paper proposed a real-time air pressure monitoring scheme based on a DSP system. We performed real-time air pressure stream data processing by leveraging the high performance, low delay, scalability, and fault tolerance of Kafka. Large-scale real-time data transmission was reliable, and the dynamic needs of real-time air pressure monitoring were met. Additionally, we also designed an intuitive and user-friendly visualization interface for data analysis and decision making. Simultaneously, the scheme processes the data gathered by sensors according to preset rules and issues alerts for potential anomalies, providing users with real-time, accurate, and reliable services for monitoring air pressure stream data in various environments. In future research, we plan to use machine learning to process and analyze historical data. We will formulate more appropriate data processing rules for different working environments to more accurately and promptly detect anomalies in the environment.

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

Funding: This work is supported by the National Natural Science Foundation of China (grant no. 62102190).

Institutional Review Board Statement: Not applicable.

Informed Consent Statement: Not applicable.

Data Availability Statement: All data are contained within the article.

Acknowledgments: The authors thank Dehao Guo and Tao Lu for their technical research and development contributions to this project.

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

References

1. Fan, C.; Zan, C.; Zhang, Q.; Shi, L.; Hao, Q.; Jiang, H.; Wei, F. Air injection for enhanced oil recovery: In situ monitoring the low-temperature oxidation of oil through thermogravimetry/differential scanning calorimetry and pressure differential scanning calorimetry. *Ind. Eng. Chem. Res.* **2015**, *54*, 6634–6640. [[CrossRef](#)]
2. Mohindru, P. A REVIEW ON SMART SENSORS USED IN CHEMICAL INDUSTRY 4.0. *J. Data Acquis. Process.* **2023**, *38*, 1172.
3. Caban, J.; Drożdźiel, P.; Barta, D.; Liščák, Š. Vehicle tire pressure monitoring systems. *Diagnostyka* **2014**, *15*, 11–14.
4. Yang, S.; Zhang, X.; Liang, J.; Xu, N. Research on Optimization of Monitoring Nodes Based on the Entropy Weight Method for Underground Mining Ventilation. *Sustainability* **2023**, *15*, 14749. [[CrossRef](#)]
5. Folgado, F.J.; Calderón, D.; González, I.; Calderón, A.J. Review of Industry 4.0 from the Perspective of Automation and Supervision Systems: Definitions, Architectures and Recent Trends. *Electronics* **2024**, *13*, 782. [[CrossRef](#)]
6. Domínguez-Bolaño, T.; Campos, O.; Barral, V.; Escudero, C.J.; García-Naya, J.A. An overview of IoT architectures, technologies, and existing open-source projects. *Internet Things* **2022**, *20*, 100626. [[CrossRef](#)]
7. Fortoul-Diaz, J.A.; Carrillo-Martinez, L.A.; Centeno-Tellez, A.; Cortes-Santacruz, F.; Olmos-Pineda, I.; Flores-Quintero, R.R. A Smart Factory Architecture Based on Industry 4.0 Technologies: Open-Source Software Implementation. *IEEE Access* **2023**, *11*, 101727–101749. [[CrossRef](#)]

8. Kalsoom, T.; Ramzan, N.; Ahmed, S.; Ur-Rehman, M. Advances in sensor technologies in the era of smart factory and industry 4.0. *Sensors* **2020**, *20*, 6783. [[CrossRef](#)] [[PubMed](#)]
9. Howard, J.; Murashov, V.; Cauda, E.; Snawder, J. Advanced sensor technologies and the future of work. *Am. J. Ind. Med.* **2022**, *65*, 3–11. [[CrossRef](#)]
10. Waworundeng, J.M.S.; Tiwow, D.F.; Tulangi, L.M. Air pressure detection system on motorized vehicle tires based on iot platform. In Proceedings of the 2019 1st International Conference on Cybernetics and Intelligent System (ICORIS), Medan, Indonesia, 8–9 October 2022; IEEE: Piscataway, NJ, USA, 2019; Volume 1, pp. 251–256.
11. Fay, C.D.; Healy, J.P.; Diamond, D. Advanced IoT Pressure Monitoring System for Real-Time Landfill Gas Management. *Sensors* **2023**, *23*, 7574. [[CrossRef](#)]
12. Hassan, M.N.; Islam, M.R.; Faisal, F.; Semantha, F.H.; Siddique, A.H.; Hasan, M. An IoT based environment monitoring system. In Proceedings of the 2020 3rd International Conference on Intelligent Sustainable Systems (ICISS), Thoothukudi, India, 3–5 December 2020; IEEE: Piscataway, NJ, USA, 2020; pp. 1119–1124.
13. Akanbi, A. Estemd: A distributed processing framework for environmental monitoring based on apache kafka streaming engine. In Proceedings of the 4th International Conference on Big Data Research, Tokyo, Japan, 27–29 November 2020; pp. 18–25.
14. Chen, Z.; Kim, M.; Cui, Y. SaaS application mashup based on High Speed Message Processing. *KSII Trans. Internet Inf. Syst. (TIIS)* **2022**, *16*, 1446–1465.
15. Akanbi, A.; Masinde, M. A distributed stream processing middleware framework for real-time analysis of heterogeneous data on big data platform: Case of environmental monitoring. *Sensors* **2020**, *20*, 3166. [[CrossRef](#)]
16. Costin, A.T.; Zinca, D.; Dobrota, V. A Real-Time Streaming System for Customized Network Traffic Capture. *Sensors* **2023**, *23*, 6467. [[CrossRef](#)] [[PubMed](#)]
17. Alam, M.; Tehranipoor, M.M.; Guin, U. TSensors vision, infrastructure and security challenges in trillion sensor era: Current trends and future directions. *J. Hardw. Syst. Secur.* **2017**, *1*, 311–327. [[CrossRef](#)]
18. Lee, R.; Zhang, M.; Yan, J. Research on IIoT Cloud-Edge Collaborative Stream Processing Architecture for Intelligent Factory. In Proceedings of the 2023 IEEE Smart World Congress (SWC), Portsmouth, UK, 28–31 August 2023; IEEE: Piscataway, NJ, USA, 2023; pp. 711–716.
19. Demers, A.J.; Gehrke, J.; Panda, B.; Riedewald, M.; Sharma, V.; White, W.M. Cayuga: A General Purpose Event Monitoring System. In Proceedings of the Cidr, Asilomar, CA, USA, 7–10 January 2007; Volume 7, pp. 412–422.
20. Hsieh, C.C.; Hsieh, Y.C. Reliability and cost optimization in distributed computing systems. *Comput. Oper. Res.* **2003**, *30*, 1103–1119. [[CrossRef](#)]
21. Kejariwal, A.; Kulkarni, S.; Ramasamy, K. Real time analytics: Algorithms and systems. *arXiv* **2017**, arXiv:1708.02621.
22. Lara, R.; Benitez, D.; Caamano, A.; Zennaro, M.; Rojo-Alvarez, J.L. On real-time performance evaluation of volcano-monitoring systems with wireless sensor networks. *IEEE Sens. J.* **2015**, *15*, 3514–3523. [[CrossRef](#)]
23. Isah, H.; Abughofa, T.; Mahfuz, S.; Ajerla, D.; Zulkernine, F.; Khan, S. A survey of distributed data stream processing frameworks. *IEEE Access* **2019**, *7*, 154300–154316. [[CrossRef](#)]
24. Pajarola, R. Stream-processing points. In Proceedings of the VIS 05. IEEE Visualization, Minneapolis, MN, USA, 23–28 October 2005; IEEE: Piscataway, NJ, USA, 2005; pp. 239–246.
25. Karimov, J.; Rabl, T.; Katsifodimos, A.; Samarev, R.; Heiskanen, H.; Markl, V. Benchmarking distributed stream data processing systems. In Proceedings of the 2018 IEEE 34th International Conference on Data Engineering (ICDE), Paris, France, 16–19 April 2018; IEEE: Piscataway, NJ, USA, 2018; pp. 1507–1518.
26. Carvalho, O.; Roloff, E.; Navaux, P.O. A distributed stream processing based architecture for IoT smart grids monitoring. In Proceedings of the Companion Proceedings of the 10th International Conference on Utility and Cloud Computing, Austin, TX, USA, 5–8 December 2017; pp. 9–14.
27. Jafarpour, H.; Desai, R.; Guy, D. KSQL: Streaming SQL Engine for Apache Kafka. In Proceedings of the EDBT, 22nd International Conference on Extending Database Technology, Lisbon, Portugal, 26–29 March 2019; pp. 524–533.
28. Wu, H.; Shang, Z.; Wolter, K. Performance prediction for the apache kafka messaging system. In Proceedings of the 2019 IEEE 21st International Conference on High Performance Computing and Communications; IEEE 17th International Conference on Smart City; IEEE 5th International Conference on Data Science and Systems (HPCC/SmartCity/DSS), Zhangjiajie, China, 10–12 August 2019; IEEE: Piscataway, NJ, USA, 2019; pp. 154–161.
29. Singh, B.; Chaitra, B. Comprehensive Review of Stream Processing Tools. *Int. Res. J. Eng. Technol.* **2020**, *7*, 3537–3540.
30. Sanjana, N.; Raj, S.; Sandhya, S. Real-time Event Streaming for Financial Enterprise System with Kafka. In Proceedings of the 2023 3rd Asian Conference on Innovation in Technology (ASIANCON), Pune, India, 25–27 August 2023; IEEE: Piscataway, NJ, USA, 2023; pp. 1–6.
31. D’Ausilio, A. Arduino: A low-cost multipurpose lab equipment. *Behav. Res. Methods* **2012**, *44*, 305–313. [[CrossRef](#)]
32. Mahmood, K.; Orsborn, K.; Risch, T. Wrapping a nosql datastore for stream analytics. In Proceedings of the 2020 IEEE 21st International Conference on Information Reuse and Integration for Data Science (IRI), Las Vegas, NV, USA, 11–13 August 2020; IEEE: Piscataway, NJ, USA, 2020; pp. 301–305.
33. Györödi, C.A.; Dumșe-Burescu, D.V.; Zmaranda, D.R.; Györödi, R.Ş. A comparative study of MongoDB and document-based MySQL for big data application data management. *Big Data Cogn. Comput.* **2022**, *6*, 49. [[CrossRef](#)]

34. Györödi, C.; Györödi, R.; Pecherle, G.; Olah, A. A comparative study: MongoDB vs. MySQL. In Proceedings of the 2015 13th International Conference on Engineering of Modern Electric Systems (EMES), Oradea, Romania, 11–12 June 2015; IEEE: Piscataway, NJ, USA, 2015; pp. 1–6.
35. Raptis, T.P.; Cicconetti, C.; Passarella, A. Efficient topic partitioning of Apache Kafka for high-reliability real-time data streaming applications. *Future Gener. Comput. Syst.* **2024**, *154*, 173–188. [[CrossRef](#)]
36. Fu, G.; Zhang, Y.; Yu, G. A fair comparison of message queuing systems. *IEEE Access* **2020**, *9*, 421–432. [[CrossRef](#)]
37. D’silva, G.M.; Khan, A.; Bari, S. Real-time processing of IoT events with historic data using Apache Kafka and Apache Spark with dashing framework. In Proceedings of the 2017 2nd IEEE International Conference on Recent Trends in Electronics, Information & Communication Technology (RTEICT), Bangalore, India, 19–20 May 2017; IEEE: Piscataway, NJ, USA, 2017; pp. 1804–1809.
38. Scott, D.; Gamov, V.; Klein, D. *Kafka in Action*; Simon and Schuster: New York, NY, USA, 2022.
39. Apache Kafka. Available online: <https://kafka.apache.org> (accessed on 11 April 2024).
40. Buzzoni, E.; Forlani, F.; Giannelli, C.; Mazzotti, M.; Parisotto, S.; Pomponio, A.; Stefanelli, C. The advent of the internet of things in airfield lightning systems: Paving the way from a legacy environment to an open world. *Sensors* **2019**, *19*, 4724. [[CrossRef](#)] [[PubMed](#)]
41. Happ, D.; Karowski, N.; Menzel, T.; Handziski, V.; Wolisz, A. Meeting IoT platform requirements with open pub/sub solutions. *Ann. Telecommun.* **2017**, *72*, 41–52. [[CrossRef](#)]
42. Berlian, M.H.; Sahputra, T.E.R.; Ardi, B.J.W.; Dzatmika, L.W.; Besari, A.R.A.; Sudiby, R.W.; Sukaridhoto, S. Design and implementation of smart environment monitoring and analytics in real-time system framework based on internet of underwater things and big data. In Proceedings of the 2016 International Electronics Symposium (IES), Denpasar, Indonesia, 29–30 September 2016; IEEE: Piscataway, NJ, USA, 2016; pp. 403–408.
43. Buddhika, T.; Pallickara, S. Neptune: Real time stream processing for internet of things and sensing environments. In Proceedings of the 2016 IEEE International Parallel and Distributed Processing Symposium (IPDPS), Chicago, IL, USA, 23–27 May 2016; IEEE: Piscataway, NJ, USA, 2016; pp. 1143–1152.
44. Han, S.; Gong, T.; Nixon, M.; Rotvold, E.; Lam, K.Y.; Ramamritham, K. Rt-dap: A real-time data analytics platform for large-scale industrial process monitoring and control. In Proceedings of the 2018 IEEE International Conference on Industrial Internet (ICII), Seattle, WA, USA, 21–23 October 2018; IEEE: Piscataway, NJ, USA, 2018; pp. 59–68.
45. Corral-Plaza, D.; Medina-Bulo, I.; Ortiz, G.; Boubeta-Puig, J. A stream processing architecture for heterogeneous data sources in the Internet of Things. *Comput. Stand. Interfaces* **2020**, *70*, 103426. [[CrossRef](#)]
46. Pallickara, S.; Ekanayake, J.; Fox, G. Granules: A lightweight, streaming runtime for cloud computing with support, for map-reduce. In Proceedings of the 2009 IEEE International Conference on Cluster Computing and Workshops, New Orleans, LA, USA, 31 August–4 September 2009; IEEE: Piscataway, NJ, USA, 2009; pp. 1–10.

Disclaimer/Publisher’s Note: The statements, opinions and data contained in all publications are solely those of the individual author(s) and contributor(s) and not of MDPI and/or the editor(s). MDPI and/or the editor(s) disclaim responsibility for any injury to people or property resulting from any ideas, methods, instructions or products referred to in the content.