

Received 21 July 2023, accepted 8 August 2023, date of publication 15 August 2023, date of current version 8 September 2023.

Digital Object Identifier 10.1109/ACCESS.2023.3305586

## RESEARCH ARTICLE

# A Study on Big Data Collecting and Utilizing Smart Factory Based Grid Networking Big Data Using Apache Kafka

SANGIL PARK<sup>1</sup>, (Member, IEEE), AND JUN-HO HUH<sup>2,3</sup>, (Member, IEEE)

<sup>1</sup>MYLINK Inc., Seoul 04533, Republic of Korea

<sup>2</sup>Department of Data Science, National Korea Maritime and Ocean University, Yeongdo-gu, Busan 49112, Republic of Korea

<sup>3</sup>Interdisciplinary Major of Ocean Renewable Energy Engineering, National Korea Maritime and Ocean University, Busan 49112, Republic of Korea

Corresponding author: Jun-Ho Huh (72networks@kmou.ac.kr)

This work was supported by the Research Promotion Program through the Korea Maritime and Ocean University Research Fund, in 2022.

**ABSTRACT** In the Smart Factory environment of the 4<sup>th</sup> industrial revolution, much data is generated from equipment, IoT sensors, and a wide range of manufacturing systems. As manufacturing sites are scattered around the world, information exchange between geographically remote factories is ever more necessary. Also, higher quality and effective management can be achieved by integrating and analyzing the collected and refined data and deriving organic results in the ever-rapidly changing manufacturing environment. However, as the main factory consists of a separate network with much data generated, it is highly difficult to gather all data into one and refine it. The most widely used method of data gathering at present has an architecture where data is linked through integration of the centrally configured solutions for data gathering and linkage. In other words, legacy systems most commonly used in manufacturing sites such as ERP, MES, WMS, etc. use the central system called ESB or EAI, to collect data with the SOA method for inter-system data linkage and collection and pass it on to another legacy system. The centralized method is not suitable for gathering and converging data generated from dozens or hundreds of different factories that are regionally dispersed or made up of independent networks and are also extremely vulnerable in terms of security and safety. This article aims to investigate how to stably and effectively exchange and collect data in geographically remote, independent networks using Apache Kafka, one of the big data ecosystems, and how to enable easy analysis of such data so that users can effectively utilize it.

**INDEX TERMS** Smart factory, Apache Kafka, ESB, EAI, data link, elastic search, Zookeeper, grid network.


## I. INTRODUCTION

The entire world is going through an unprecedented environment, namely, COVID-19. The non-face-to-face operations due to COVID-19 have led to increased rates of teleworking, and social change such as work-life balance has also led to reduced working hours. Furthermore, as global companies increase in number, geographical boundaries are beginning to disappear. Amidst the changes in working hours and environment, there arose a need for a method that enables quick and effective work performance. In order to solve the problems outlined above, companies have introduced a number of

business systems such as ERP, HR, etc., and the systems have even been distributed due to globalization.

While manufacturers continue to make further use of solutions in smart factories and non-face-to-face environments, information is rather more distributed, decreasing work efficiency. Efforts are made to solve this problem and find ways to integrate the information into one for use and to effectively use diverse systems, and a demand for a system that accurately and quickly notifies the tasks at hand, rather than a complex and difficult system, is higher than ever before. At present, each system separately notifies the tasks; and only some systems are equipped with such a function.

If a work notification is necessary, additional development and cost are incurred for each system, and the function is not

The associate editor coordinating the review of this manuscript and approving it for publication was Chong Leong Gan .

a single, integrated function; rather, it is limited to certain systems.

If a system is distributed for separate management of information, much information that is not refined is generated, which makes its utilization difficult. Or, the resulting unnecessary data makes it difficult for customers (users) to accurately identify which task to work on, and how to proceed with it, greatly lowering work efficiency.

Smart Factory manufacturers make use of an in-house network system for security reasons, and a wide variety of data are generated in bulk, in real-time. However, there are limits to big data processing with existing data link systems only, such as EAI, ESB, etc. In addition, in the manufacturing industry, there is a great variety of system types, as well as data types. If each system has to be modified and developed for data gathering and utilization for each of these conditions, much cost is expected to be incurred. A method to refine data, and store it in multiple databases to enable indexing and AI analysis for quick data extraction is also necessary, where collected data, not a simple data linkage, is needed.

Therefore, this paper intends to examine a method to quickly transmit user-required information, facilitate system linkage and quickly extract (collect) and utilize the gathered information by gathering various types of data on manufacturing systems distributed according to regions (global, regional smart factory) or purpose, using various means.

## II. BACKGROUND KNOWLEDGE

As for the Big Data collection and analysis method, Bajer [1] studied building, searching, and visualizing a data hub where various types of data including IoT data are stored, using Elasticsearch, Logstash, and Kibana (ELK), and Choi, Bomin et al. [2] used NoSQL-based MapReduce to collect information for effective firewall log analysis.

In relation to Apache Kafka and Big Data collection, Hiranman et al. [3] examined the stream data processing of Apache Kafka and how effective Kafka's high performance is for big data stream processing, and Shree et al. [4] showed that Kafka performs well and is effective for big data analysis and processing. Bhole Rahul Hiranman, using the advantage of its scalability, distribution and capability of high processing through stable results, checked how Apache Kafka works in big data stream processing and found that it can process higher amount than existing messaging systems. Shree et al. [4], in *Kafka: The Modern Platform for Data Management and Analysis in Big Data Domain*, examined the performance evaluation and effectiveness of Apache Kafka, and various ways to bring data between systems and applications and real-time streaming.

Meanwhile, in *Improvement of Apache Kafka Streaming Using Partition and Multi-Threading in Big Data Environment*, Leang et al. [5] used Hadoop and HBase to handle large data in a manufacturing environment and used Apache Kafka as a data streaming pipeline. Also, Apache Spark, with an Apache Kafka interface, enabled real-time data processing and analysis. Encryption was performed in a manner

that includes a public key and a private key. Through the aforementioned studies, it has been proven to increase the performance and accuracy of data storage, processing, and security in the manufacturing environment.

With regard to big data collection, Xu et al. [6] looked into the current status of the collection of industrial big data generated in the Industry 4.0 environment, ontology-based modeling, prognosis based on industrial big data, AI learning of equipment, etc. Noac'H et al. [7] researched how the collection performance can affect the entire stream processing through performance evaluation of Apache Kafka, and showed which element has the greatest impact by measuring various elements. Bandi et al. [8] collected big data streaming generated from mobile devices and IoT devices using the Kafka technology based on the Kappa architecture and transmitted the tableau in real time using Rockset to examine a method for visualization. Data was collected from Twitter at 30-second intervals, using Twitter API. Bandi et al. [8] linked data using Rockset as middleware and used the Kappa architecture as the prerequisite for streaming data. However, it differs from this study in that it collected and visualized data using Apache Kafka based on a single source.

On the other hand, Lavanya et al. [9] integrated technologies such as Apache Kafka, Spark, Mongo DB, and LSTM for data collection, in order to effectively forecast the streaming weather in real-time. Yang et al. [10] investigated the data collection technology of the smart grid and explained the diverse effects of energy through the pre-processing and analysis of power-related data. Such communication using power lines is considered as the agent of this study that views it as one of the methods for data collection in areas where the Internet is not active if not for power data, and as the topic for new data collection using power grid through the improved grid network collection. Moreover, Dhupa et al. [11] researched how to effectively utilize the Smart Grid through AI comparative analysis with the smart grid. Ansari et al. [12] also examined the real-time anomaly detection framework based on smart meter data collected from the smart grid big data. As for the architecture of the study, data was collected through queues using Mongo DB, Cassandra, elastic, and Hadoop, and for real-time processing and analysis, Spark was used. In addition, Apache Kafka was utilized for data linkage between Spark and the big data framework.

Leang et al. [5] looked into the storage and security of big data transmission using Apache Kafka and Spark in the manufacturing environment. And Sahal et al. [18] compared the open source functions including Kafka to collect and stream process big data for prediction maintenance and repair, and proposed, using cases, the optimal combination of big data technologies. They differ by industry, but largely proposed three architectures: Apache Kafka, Amazon Kines, and RabbitMQ in relation to data collection queues.

### A. APACHE KAFKA

Apache Kafka, an open-source distributed message processing system developed by the Apache Software Foundation

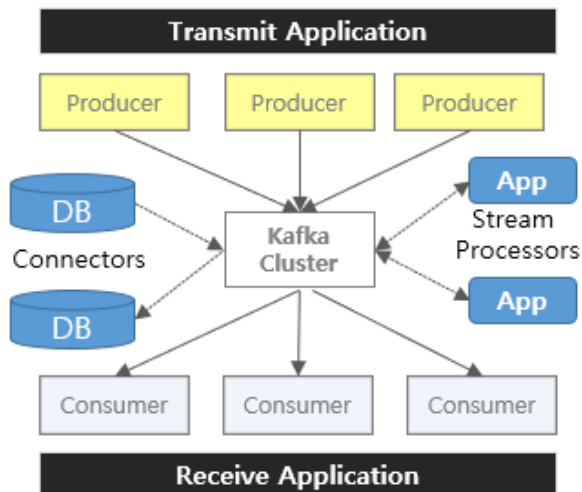


FIGURE 1. The architecture of Apache Kafka.

and a type of MOM (Message Oriented Middleware) software, asynchronously relays message data generated in bulk for real-time processing [14]. Apache Kafka is specialized for real-time processing of large-capacity messages and is suitable for scaling up the system as it is designed based on the distributed system [1], [2]. Figure 1 shows the architecture of Apache Kafka.

Apache Kafka, an architecture that stably transmits data to the target system while buffering the intermediate data in the event of large-scale transaction data from the source system that provides data, is capable of both data collection and transmission, depending on the utilization. Also, Apache Kafka operates based on the publish-subscribe model and is made up of producers, consumers, and brokers [3], [4]. Unlike existing message processing systems where Broker directly pushes messages to the Consumer, in Apache Kafka, Consumer directly pulls the needed messages from Broker, resulting in optimal performance. Apache Kafka guarantees data permanence as it stores messages in a file format and is advantageous in that it causes few performance degradations in case of a large volume of messages, compared to existing message systems [14].

Meanwhile, Apache Kafka is made up of main elements such as a broker, topic, provider, consumer, etc. Topic plays the role of storage for processing of data generation and consumption in the broker. Broker, meaning a Apache Kafka server, serves to control the topic and is able to operate multiple Apache Kafka servers in one cluster. Provider plays the role of transmitting (publishing) data to a specific topic of broker and implements it in the application using the Apache Kafka library. Lastly, the Consumer plays the role of recipient of data from a specific topic of broker and implements it in the application using the Kafka library [5].

## B. ZOOKEEPER

Zookeeper is a tool to manage multiple Apache Kafka servers. It facilitates operations such as synchronization or master election using API, centralizes the information of each

application (Kafka), and provides such services as configuration management, group management naming, synchronization, and others [3]. Using Zookeeper, multiple Apache Kafka servers can be managed in distributed network environments rather than a single network, which helps the intricate configuration of networks. Because the sub-distributed application (Kafka) fails if Zookeeper fails, Zookeeper should also be configured as distributed [15].

As shown in Figure 2, Zookeeper configures multiple servers into a cluster, and distributed applications become respective clients that provide status and information as connected to Zookeeper servers. In this paper, Zookeeper itself was configured as a cluster to manage Zookeeper with a manager server for Zookeeper.

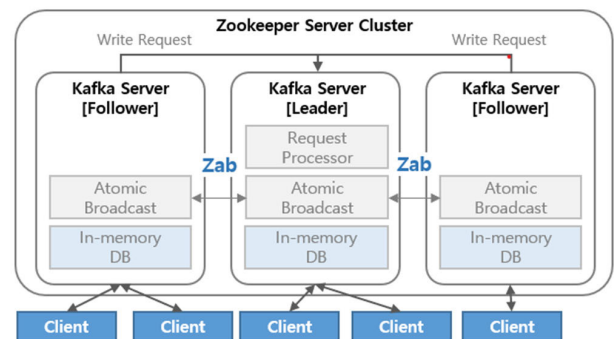


FIGURE 2. Zookeeper server cluster architecture.

## C. ELASTICSEARCH

Elasticsearch is an open-source distributed search engine developed by Shay Banon based on Apache Lucene [16]. Released in 2010 for the first time, it supports distributed search and analysis for users to search and combine various types of data such as JSON-based informal data and formal data, location information, metrics, etc. at their preference. Capable of quick and near-real-time storage, searching, and analysis of a large volume of data, it is thus used as a popular database-type search engine. Elasticsearch can configure a distributed environment in multiple PCs with a relatively simple setup. As it modifies the original data and duplicated data according to the data capacity and PC specifications, it configures a horizontally distributed environment, providing a more stable operating environment than a single server. Due to these characteristics, Elasticsearch is used independently as a search engine, but often linked with Kibana and Logstash to configure the Elastic Stack System and utilized as a user application [17].

Elasticsearch Cluster is configured as shown in Figure 3; A node is a physical server that makes up a cluster, and each shard, as a subset of the index, is made up using Lucene. It stores real data and indexes and is classified into the primary shard and replica shard. The primary shard is a basic index that makes up a shard, and a replica shard is a replica of a primary shard stored in another distributed node. It is a document type, and logical category/partition within an

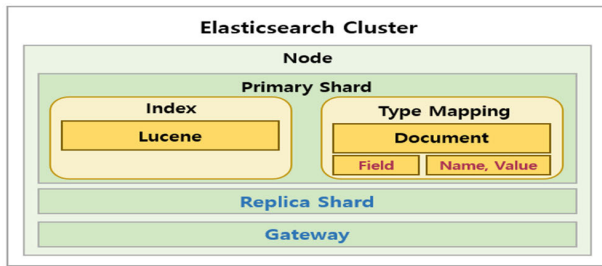


FIGURE 3. Configuration of elasticsearch cluster.

index, similar to a table in DBMS. A document is the basic unit of data storage managed in Elasticsearch and is expressed in JSON (JavaScript Object Notation). In addition, a field is an element that makes up a document and consists of a name and value. A gateway stores information such as cluster status and index setup. Since Elasticsearch is designed to facilitate horizontal scaling, in a large-capacity environment, a node may be added so that the cluster recognizes it to scale it up.

#### D. AES ENCRYPTION

The Advanced Encryption Standard (AES) is a cryptographic algorithm chosen to replace DES as a data encryption standard by the National Institute of Standards and Technology (NIST). Approved by NSA for top-secret information, this cryptographic algorithm is highly secure. It is a symmetric-key algorithm, meaning the same key is used for both encrypting and decrypting the data.

It allows different key and block lengths: 128bit, 192bit, or 256bit, and shows outstanding performance in speed and efficiency. The AES has the SPN (Substitution Permutation Network) structure - it uses the substitution layer and permutation layer to achieve confusion and diffusion. While parallel operations can be implemented, a separate decryption module should be implemented for decryption [18], [19].

### III. COLLECTING SMART FACTORY BASED GRID NETWORKING BIG DATA USING APACHE KAFKA

The prerequisites to gather various types of data (file, RDBMS, PLC (equipment), DAQ, Web, etc.) distributed according to regions (global, regional smart factory) or purpose (ERP, MES, WMS, SCM) using various means (FTP, REST API, etc.) are as follows: First, it should be made available to collect data from each data source system using the pull method without additional modification or development by taking into consideration various systems. Second, if collected data is required, data should be refined and stored in the big data DB such as Redis, Mongo DB, etc., and support should be provided to enable index analyzer (Elasticsearch) and AI analysis. Third, data should be collected, refined, and linked for manufacturing of grids (mesh-type) that process and transmit the necessary information to a specific system. Fourth, under the premise that each network differs by the network area or environment, it should be possible to collect data in the applicable network without any loss of data. Fifth,

cost reduction and stable performance should be guaranteed, and the stability-assured architecture should be configured.

Figure 4 proposed Architecture for Big Data Collection to meet the above conditions, an agent that collects data from a closed internal network, a middleware server that manages the agent and relays the data transmission and management, and a management server/monitoring server that manages Zookeeper, Apache Kafka, and Agent are required, in addition to the basic Apache Kafka configuration.

The proposed architecture for big data collection is, as shown in Figure 4, configured to collect and transmit data through the agent at the data source end and to allow distribution and agent management through the middle trans server in the middle in charge of agent management and distribution. The Apache Kafka server was distributed up to three units or more, and it was set up so that a partition is automatically generated when a data item is set. Zookeeper was used to manage Apache Kafka, and a data refining engine was added that is capable of processing the Apache Kafka message again. The refined data was selectively stored in Redis or Mongo DB in a mapped format, and the key information of the data was stored in the Maria DB. Also, it was designed to provide data if real-time analysis is necessary by linking Apache Kafka with Spark. The manager server acts as a center of the server collecting grid-type big data, such as Apache Kafka, Zookeeper, monitoring, agent, middle trans server, DB storage, data processing, etc. The data collected as such is stored in various databases according to the method set as the monitoring tool of the manager server in units of agent and topic, or if transmitted to an AI analysis program for data analysis or another legacy system, it may be utilized for various applications or to search for data stored in the analysis and index information according to settings.

Various tools for Big Data collection such as the agent, middle trans server, broker, Zookeeper, data filtering engine (RPA), Spark, etc. can be managed through the monitoring tool of the manager server, and the monitoring tool checks the status of the agent, broker, and middle trans server in real-time, and reports any anomaly to the administrator.

The consumer that uses the collected data delivers it as is or processed or analyzed results to systems such as the web, mobile application, messenger, legacy system, etc., according to settings. It also relays services, such as linking different APIs including Slack, Google Calendar, Okta, etc.

Figure 5 briefly shows the architecture for distributed data collection. The legacy zone is a separate network environment where real data is collected and used and is an individual Smart Factory Plant. The legacy zone has applications in operation such as WMS, MES, ERP, etc., and is configured to collect data generated therefrom and deliver it to another legacy zone or collect data with the same purpose generated from multiple legacy zones, perform statistical analysis with the data and deliver it to users.

In order to collect data from a closed network with a firewall as above, the agent should be installed in the network, and the agent uses node.js.

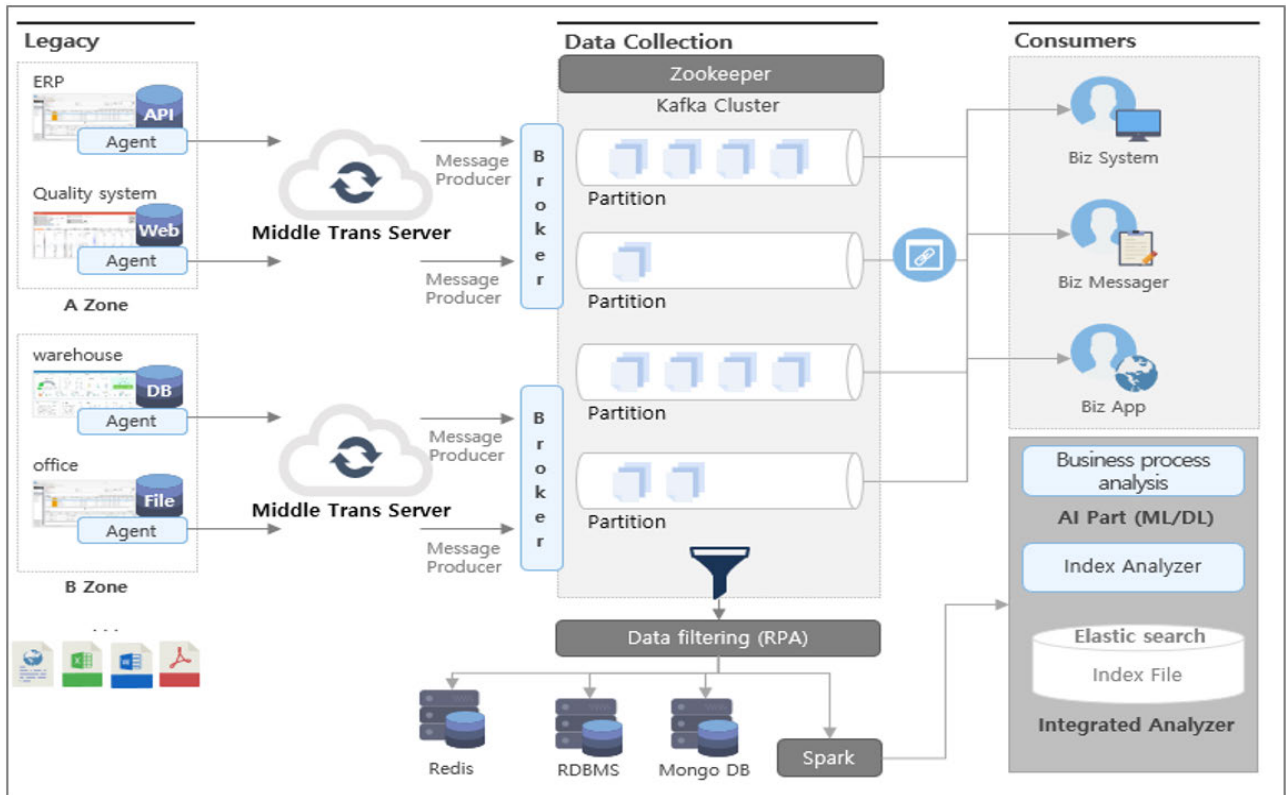


FIGURE 4. Proposed architecture for big data collection.

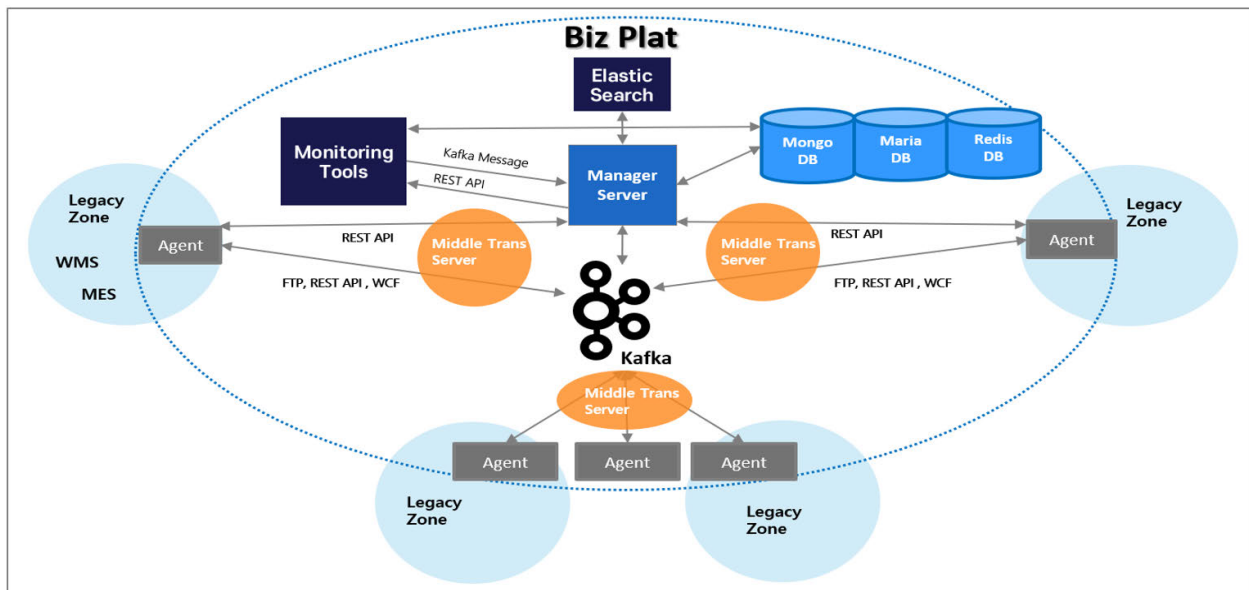


FIGURE 5. Proposed server configuration for big data collection.

As for an agent developed with node.js, there is no delay even when multiple data are simultaneously collected and delivered from one agent, as it is executed immediately without waiting for processing to be done once the I/O operation starts, thanks to the single through and non-blocking IO, which are the characteristics of node.js. However, because it

should be executed after registering the job through the event, the agent status and job schedule should be checked from the middle trans server and manager server to continuously deliver and execute events.

The agent can collect data by connecting various databases such as Oracle, Maria, MySQL, etc., transmitting files using

FTP protocol, rest API, web crawler, etc., and data meeting specific requirements may selectively be collected, such as new data, changed data, entire data, etc.

In addition, as the agent that uses node.js is very light and can be distributed in combination after separating specific functions for development, large-capacity functions, such as the rest API function, may be excluded for distribution, or a specific function may be improved to configure an agent specialized for the relevant collection section. As can be seen in Figure 6, node.js is a tool developed using a simple and light coding scheme based on javascript and can be distributed in combination after separate configurations of functions for each file. Also, when distributed, it is distributed in installable files such as exe or pkg, eliminating the risk of exposing the configured source.

TABLE 1. Sample of the interface specification.

ID	Name	From-To
ZAD01	Request DB access list for agent	Manager Agent →
ZAF02	Agent FTP connection settings	Manager Agent →
ZAP01	Agent job profile settings (schedule)	Manger Agent →
ZAB07	Prepare a list of agent-managed batches	Manger Agent →
ZMF02	Request change of server.properties file	Monitoring→ Manager
ZRE03	Zookeeper starting performance result	Agent Manger →

```

// zookeeper status information
await mongo.find('zookeeperStatus',
  { agentId: jsonData.data.result.etc.agent.id }
).then(async (zooData) => {
  console.log(zooData);
  if (zooData.length > 0) {
    var zoo = {
      serverId: zooData[0].serverId,
      batchType: zooData[0].batchType,
      batchSize: zooData[0].batchTime
    };
    var sendMsgZoo = await
      util.makeMessage("CAB04", zoo);
    sendMsgZoo.header.receiver =
      jsonData.data.result.etc.agent.id;
    await producer.sendData2(sendTopic,
      sendMsgZoo, useProducer, keyCode);
  }
});

```

FIGURE 6. Program for agent to check Zookeeper status.

The agent can collect data by connecting various databases such as Oracle, Maria, MySQL, etc., transmitting files using FTP protocol, rest API, web crawler, etc., and data meeting specific requirements may selectively be collected, such as new data, changed data, entire data, etc. The agent's event was defined as the interface ID as described in Table 1; the ID was defined in general for job performance, from the middle trans server, manager server, monitoring server, and legacy to application, as well as the agent.

The agent monitors the status of the system where it is installed at one-minute intervals, and the information can be checked in the monitoring tool as shown in Figure 7.

The agent monitors the service or program set up for management and is capable of executing a restart in the event of an anomaly in the service or program or agent failure.

Through this paper, it was possible to collect, analyze and manage the data distributed across manufacturers and systems in an intended format, and even where networks for each system that are regionally distributed are divided by the intranet, it was possible to collect the information in the intranet to the center, enabling integrated analysis of the data. Furthermore, it was possible to link data through the central solution for data sharing and collaboration between

organizations and companies and to search and utilize various heterogeneous system data in an integrated form. In addition, in practice, analyzing the cause of defects and tracking the production history in the manufacturing system were configured through Redis DB and Elastic Search Index.

For Apache Kafka management, Zookeeper was placed in higher locations, and in the testing environment, three Apache Kafka cluster servers were configured under one Zookeeper. In the real service operating environment, three zookeepers were located at another cluster network end, with three Apache Kafka servers under each Zookeeper. For clear cluster distribution, the network service ends were separated with one set configured in Naver Cloud Platform of Korea, another set in AWS, and the other in Azure, and for further distribution, the server locations were distributed across Korea, China, etc.

The information in each legacy is extracted through the connected agent, and the extracted information is delivered to the manager server (Kafka) through the middle trans server and message producer. The relevant information is partitioned through data classification to transmit the user-required information to the user's app. Also, according to the need of the gathered data, it may be stored in an unstructured database such as Oracle, RDB, Redis, or Mongo DB.

Zookeeper manages the Apache Kafka broker master, and there must be one broker master for each Apache Kafka cluster. Zookeeper also has a master and slave, and if a failure occurs in the master, the slave acts as an assistant. For further scaling up in the grid environment, an additional increase of Zookeeper is also necessary. Zookeeper controls the broker through the master broker, and if a failure occurs in the master broker, another broker becomes the master broker. The broker cluster is an assembly of brokers, and the broker delivers messages through the topic. The topic has n number of partitions and n number of replicas; in general, one partition delivers the message through one queue. However, if it is designed with a single partition as above, in the event of a message delay, the subsequent message is not delivered. As such, three or more partitions should be set up to contain the message in a

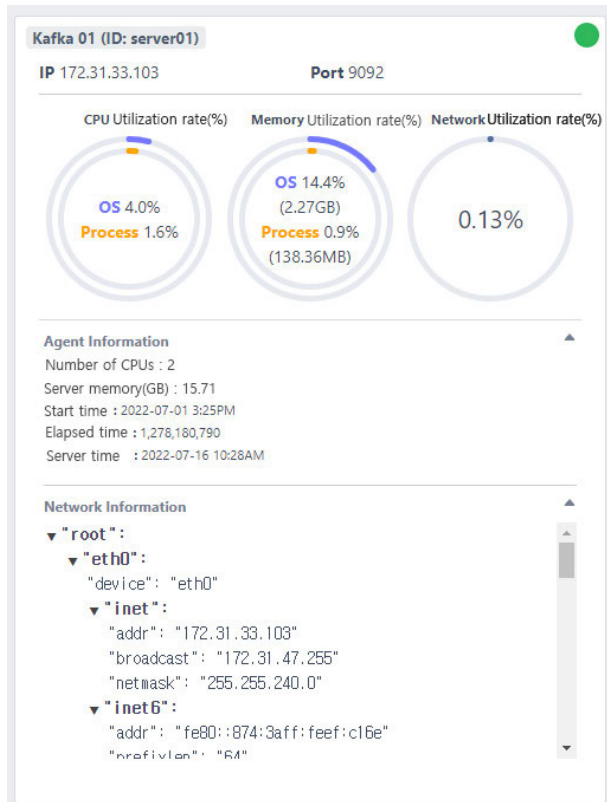


FIGURE 7. Agent management in the monitoring tool.

queue through the round-robin method to handle the issue of message delay.

The agent is installed in every Zookeeper and broker server, and the monitoring server checks the leader and follower and the usage of CPU, memory and disk according to the monitoring cycle. If an anomaly is detected, the monitoring server automatically sends a notification to the person in charge and is capable of automatic restart to prompt response to failure, depending on the settings.

Zookeeper and broker (Kafka) settings are managed with a config file such as server.properties, and the config file set up in the monitoring server may be distributed through the agent or to each Zookeeper or broker to individually or collectively change the settings.

As shown in Figure 8, the monitoring server can be used to manage the status and settings for Kafka and Zookeeper.

#### IV. BIG DATA COLLECTION MANAGEMENT SCENARIOS

In grid network Big Data collection, there are a number of different scenarios between the agent, manager server, and DB for various purposes, such as data transmission, result delivery, failure, anomaly detection, etc. The Apache Kafka cluster and middle trans server were excluded from the scenarios, because the Apache Kafka cluster serves as the basis for the scenarios to be executed, and the middle trans server is weighted toward the purpose of management, such as agent management, data relay, etc. Therefore, the Apache Kafka cluster is included in the configuration of the manager

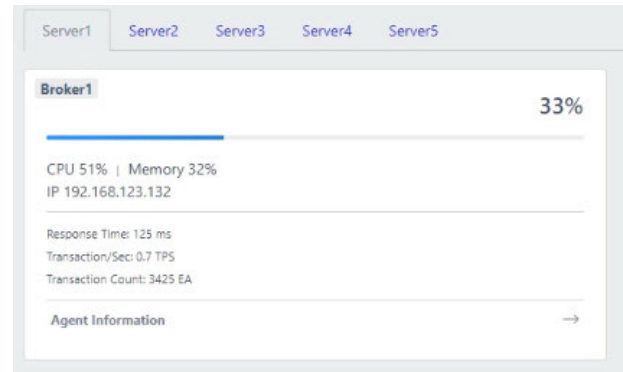


FIGURE 8. Apache Kafka cluster Broker status management UI.

server, and it is deemed that the middle trans server is located between the manager server and the agent.

Table 2 shows the communication scenarios for big data collection and processing, agent, zookeeper, broker management, and data encryption. As for Zookeeper or broker, the configure file must be changed, and the module and all functions restarted to operate with the same settings. It is highly inconvenient and difficult for users. It is almost impossible if the user is required to manage a large number of Zookeepers, brokers, and agents.

In this paper, it was configured with the web UX to store the set information in the database, and a configure file was created based on the information to change the settings or manage operations by transmission to the broker or Zookeeper. If configured as above, where an issue occurs in the settings, it is restored based on the settings information saved immediately before and allows us to accurately diagnose which setting led to the issue. All settings information is recorded in the database by revision.

The scenarios amount to 21 in number, as shown in Table 2. Processes are defined for each scenario as in Figure 9, and the processes are made up of a communication interface as in Table 1. The scenarios can be utilized to manage the entire architecture and transmit, refine, or store data.

To bring the regionally analyzed data of A company network zone (A zone) to utilize the B company network zone (B zone), the agent, as shown in Figure 10, brings the data of the database in A zone. Here, the agent within A zone must be able to access the database. To link data from A zone, the agent must be able to communicate with the middle trans server, and the middleware firewall must be open to the agent.

The communication port between the agent and middleware is designed to protect itself from intrusions via default settings for continuous change. The agent of A zone collects and transmits the data to the middle trans server by a signal due to a specific schedule or rest API communication, the middle trans server transmits the data to the manager server (Apache Kafka, Zookeeper, monitoring), and the manager server stores the data in line with the performance process or transmits it to the agent in B zone through another middle trans server. The agent in B zone stores the information in the

TABLE 2. Data collection and management scenarios.

Scenario Case	Role description
Case #1	Execute agent
Case #2	Change agent settings
Case #3	Stop and restart agent
Case #4	Execute agent DB query
Case #5	Perform file transmission
Case #6	Zookeeper status
Case #7 #8	Register/modify, delete Zookeeper
Case #9	Control Zookeeper (start, stop)
Case #10	Zookeeper settings and save zoo.cfg
Case #11	Check broker status
Case #12/ #13	Register/modify, delete broker
Case #14	Control broker (start, stop)
Case #15	Change broker settings (save and change server.properties)
Case #16	Topic list
Case #17	Sync RDBMS table / query
Case #18	Create message encryption and decryption
Case #19, #20, #21	Sync keys for message encryption and decryption (# 19: Agent), (#20: Monitor) (#21: Manager)

database or notifies the information to the legacy system of B zone through rest API, etc.

The collected data is not terminated after performing a single job of data transmission, collection, or refinement, but, if the performed process is defined, performs data processing and transmission in the defined order, and passes the completed result to the next process.

V. RESPONDING TO KAFKA CLUSTER FAILURES

For replicas, it is assumed that one cluster has three brokers, and if one broker has three partitions, the other two brokers also have replicas of the same partitions. One of the replicas is the master (leader), and the remainder are slaves (followers); if a broker failure occurs, the replica slave becomes the master. The master broker has the control over topic failures (master, slave control). If the Apache Kafka cluster is in a normal condition as shown in Figure 11, it replicates to each broker for topic A, resulting in duplicate partitions.

As can be seen in Figure 12, if a failure occurs to a broker that is not a controller among cluster brokers, the controller checks the status of other cluster brokers, and the leader partition of the failed broker is redistributed to another broker.

Also, as shown in Figure 13, the information of the newly elected broker with the leader partition is delivered to all brokers within the cluster. If a failure occurs in the controller broker, a new controller is elected as set up in Zookeeper.

One broker out of multiple cluster brokers acts as a coordinator, and if a failure occurs in the consumer group, the coordinator checks the status of the consumer group, and the partition assigned to the failed consumer is redistributed and

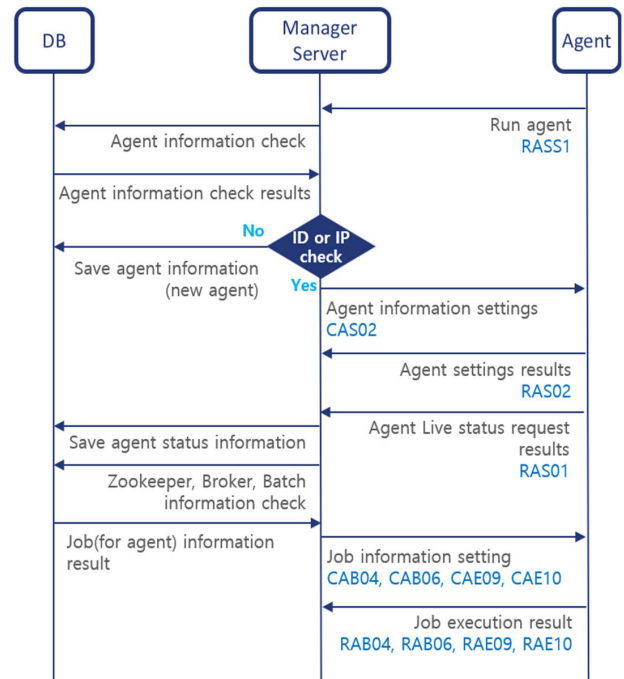


FIGURE 9. Agent execution scenarios.

assigned to another normally-functioning consumer within the same consumer group.

VI. DATA SECURITY IN GRID CLUSTERS

The Apache Kafka grid network architecture for big data collection uses AES encryption. For security, the AES encryption is used to encrypt and decrypt the key, and the randomly generated security code is additionally inserted into the encryption details thereafter. During decryption, use the key to additionally check the security code even after decryption.

The architecture has the message structure and code defined between systems. Thus, if the structure and code do not match, it detects that the delivered message is deformed [20].

Encryption and decryption generate random 16-digit keys and codes, and the manager server generates the key and distributes it to all agents and manager servers at a specific cycle. If the manager server, agent, middle trans server, and monitoring server restart, the manager server provides the key and code. If the manager server restarts and there is no key or code managed, a new code is issued.

Generating and managing the encryption keys are performed on the manager server; the manager server transmits the key to the agent and monitors the server, as shown in Figure 14. Between manager servers that are distributed, the manager server, before generating a key, requests the key value to another manager server; if there is no key value, a new key and code are generated.

If there remains a key or code due to synchronization issues, the outdated key or code is destroyed and a new key or code is transmitted to all remaining agents, manager servers, and monitor servers. With reference key and code values



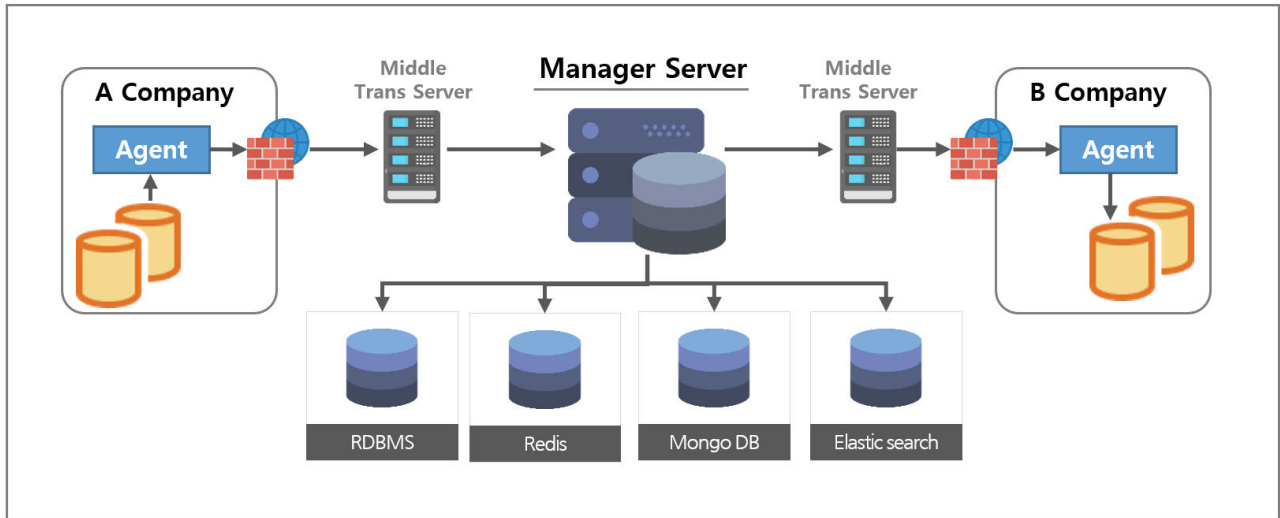


FIGURE 10. Agent execution scenarios data transmission process between network zones.

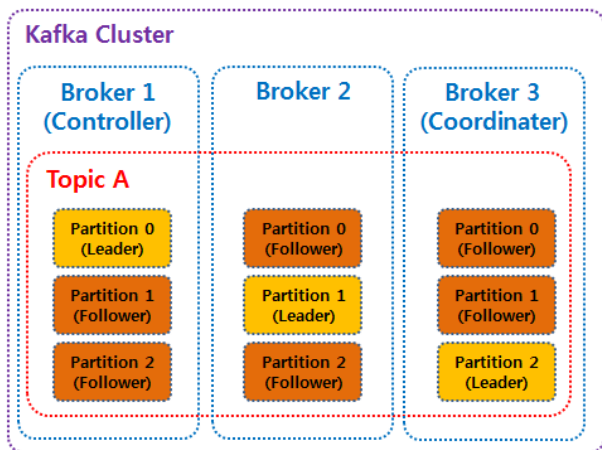


FIGURE 11. Kafka cluster in normal condition.

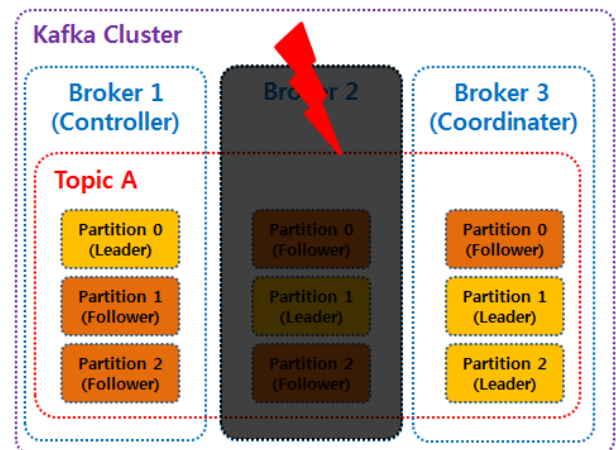


FIGURE 13. Redistribution of leader (master) partition.

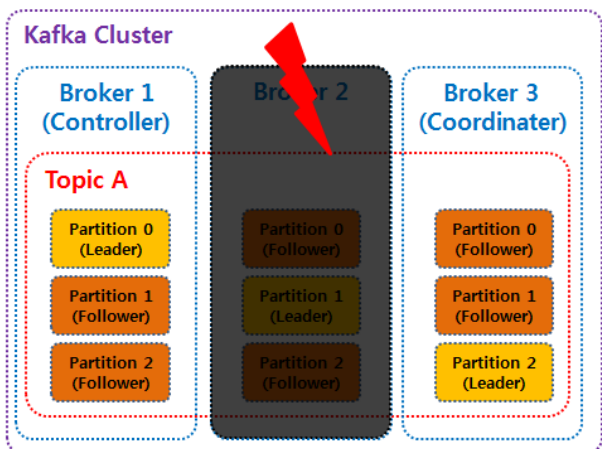


FIGURE 12. Broker failure occurred.

existing across all manager servers, where a request for a key or code is made, the encrypted key or code is requested.

### VII. BIG DATA COLLECTION CLUSTER PERFORMANCE ASSESSMENT

The Kafka grid network architecture for big data collection evaluated the server performance and transmission speed of the agent and manager server by the number of data transmissions. Since the middle trans server had only a few servers for performance evaluation, it was integrated into the manager server for evaluation. The monitoring server, too, was excluded from performance evaluation, as it had an insignificant impact on performance with its function of transmitting simple UI processing information to the manager server.

The AWS was utilized for the performance evaluation server, and the detailed specifications are shown in Table 3.

For performance evaluation, a functional integrity evaluation was first performed; the evaluation was carried out at an error-free state with a functional integrity evaluation score of 100%. NMON was used as the tool for

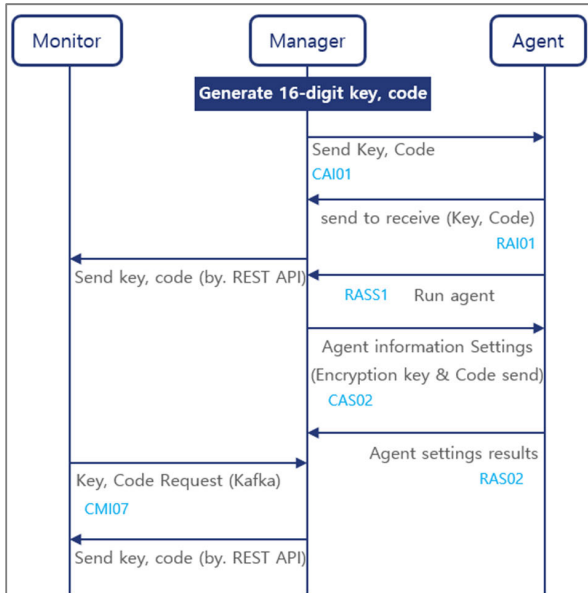


FIGURE 14. AES encryption scenarios.

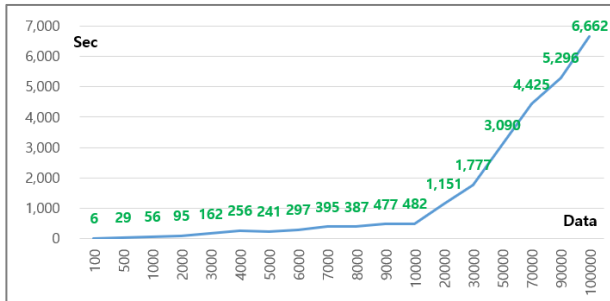


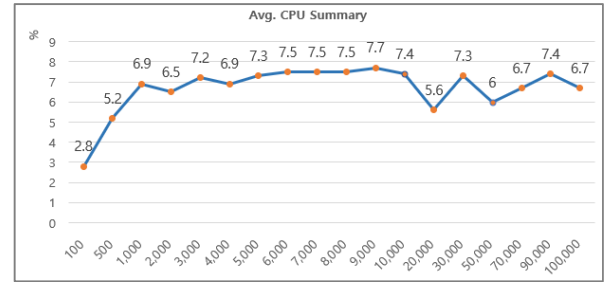
FIGURE 15. Agent stress load test.

TABLE 3. Specifications of performance evaluation server.

Server	Spec
Zookeeper & Kafka	t4g.2xlarge, 8Coe, 32GB RAM
Manager Server	Amazon t4g.2xlarge, 8 Core, 32GB RAM
Monitoring Server	Amazon EC2, c6g.2xlarge, 8 Core, 16GB RAM
FTP & RDBMS	Amazon t4g.2xlarge, 8 Core, 32GB RAM
Mongo DB	Amazon Document DB, r5.large

performance evaluation; NMON was installed in every server for performance analysis. For performance evaluation, one unit of agent, broker, monitoring, and manager server were used. As the network section is easily affected by the section state and external factors, the transmission speed, load, etc. of the network section were excluded.

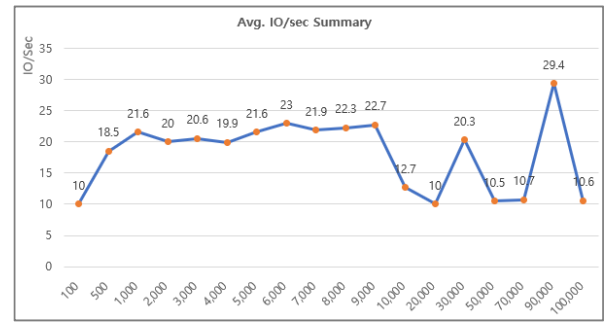
The stress load test result of the agent is shown in Figure 15. It refers to the number of data processing cases repeatedly performed by the agent upon receiving performance signals. The average size of data was 10MB, and data processing of 100 cases means the time taken to process data with a capacity of 1GB in total.



(a)



(b)



(c)

FIGURE 16. Performance evaluation: (a) Agent Stress Load Test; (b) Agent Free Memory Capacity; and (c) Agent I/O Average Processing Time.

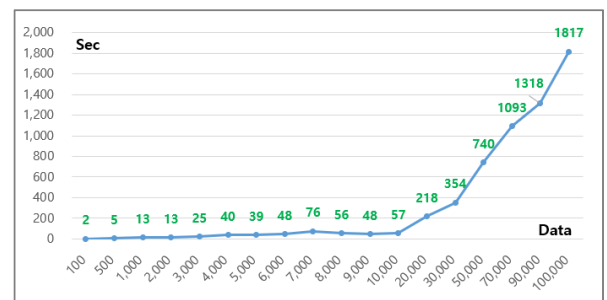
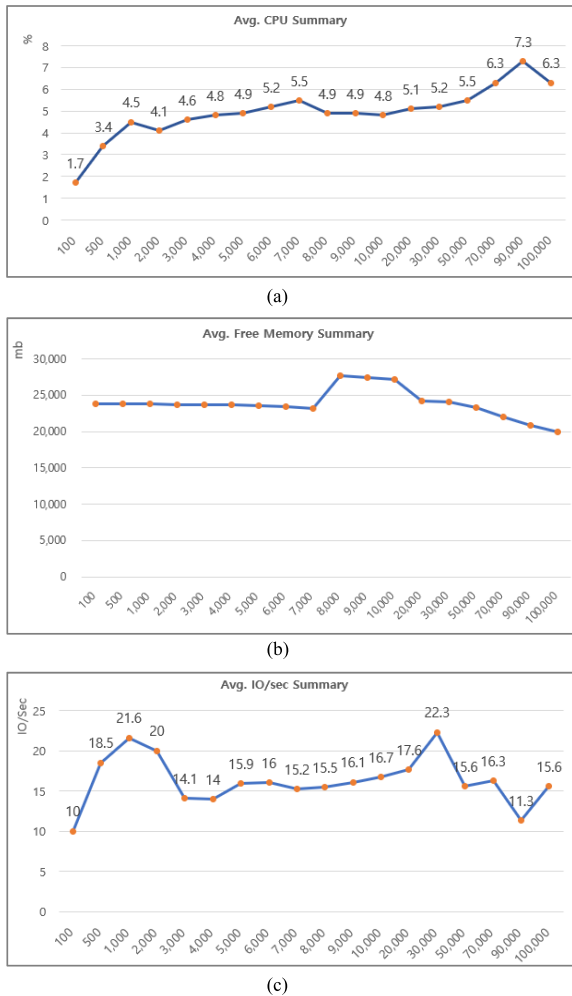


FIGURE 17. Manager server stress load test.

The agent server load generated due to data processing of 100 to 100,000 cases showed the average CPU usage in Figure 16 (a), free memory capacity in Figure 16 (b), and I/O processing speed in Figure 16 (c); the CPU was less than 10%, and the memory and I/O also appeared stable up to 20,000 cases. There is little difference in performance up to 200GB data processing, based on the average 10MB data. In other words, although the transmission time may differ depending on the network performance, the optimal volume a single



**FIGURE 18. Performance load of the manager server: (a) Manager Server CPU Average Usage Rate; (b) Manager Server Free Memory Capacity; and (c) Manager Server I/O Average Processing Time.**

agent can handle on a single occasion is 200GB, which take 19 minutes to process. Jobs such as described above seem suitable for batch jobs, and as the capacity of most real-time data does not exceed 1GB, it may be processed at a very high speed.

The manager server was evaluated up to the storage of data received from the agent in the Mongo DB. Since the Mongo DB is separate, the load due to DB storage was excluded from this evaluation.

The stress load of the manager server consistently increases as shown in Figure 17, which also shows the increase of processing time between 10,000 and 20,000 cases. As agents processed by a single manager server are as few as 40 and as many as 400 or even 500, assuming that the agent simultaneously processes data and that the manager server processes the distributed load, the data capable of real-time processing amount to about 20 MB and 500 units. However, as the number of agent servers is associated with the delay of the network section, we cannot determine with performance alone.

As shown in Figures 18(a), 18(b), and 18(c), the performance load of the manager server itself does not significantly deviate from the average usage rate or capacity. It seems that, when a load occurs in the manager server, the processing is divided, resulting in an insignificant impact on the server. For memory, however, it is advised to secure a sufficient amount of free memory from the beginning, considering the nature of data processing; it is also recommended to use a fast I/O medium for I/O.

## VIII. CONCLUSION

The Big Data collection using Apache Kafka enabled the collection of distributed data through the scalability of the agent and broker. It was therefore possible to prepare a foundation for analysis of various data. It also enabled data collection, as well as data linkage and connected analysis.

With the increasing demand for big data collection and AI analysis by organizations such as companies, public agencies, and schools, a need emerged for a system to collect and manage the data. However, it was a great challenge to extract data from legacy systems and refine it. Therefore, this paper utilized Apache Kafka, Agent, Index Analyzer, Elastic Search, etc. to propose a function to collect and extract data. The functions proposed in this article can be utilized by companies and organizations for diverse purposes including data extraction, and it is expected that a system compatible with RDMS, big data DB, etc. will be developed in the future. For the upcoming research, studies are conducted on the big data network capable of large-capacity high-speed communication, which may be used in combination with the global grid Big Data network.

## REFERENCES

- [1] M. Bajer, "Building an IoT data hub with Elasticsearch, Logstash and Kibana," in *Proc. 5th Int. Conf. Future Internet Things Cloud Workshops (FiCloudW)*, Aug. 2017, pp. 63–68.
- [2] B. Choi, J.-H. Kong, S.-S. Hong, and M.-M. Han, "The method of analyzing Firewall log data using MapReduce based on NoSQL," *J. Korea Inst. Inf. Secur. Cryptol.*, vol. 23, no. 4, pp. 667–677, Aug. 2013.
- [3] B. R. Hiranman, C. Viresh M., and K. Abhijeet C., "A study of Apache Kafka in big data stream processing," in *Proc. Int. Conf. Inf., Commun., Eng. Technol. (ICICET)*, Aug. 2018, pp. 1–3.
- [4] R. Shree, T. Choudhury, S. C. Gupta, and P. Kumar, "KAFKA: The modern platform for data management and analysis in big data domain," in *Proc. 2nd Int. Conf. Telecommun. Netw. (TEL-NET)*, Aug. 2017, pp. 1–5.
- [5] B. Leang, S. Ean, G.-A. Ryu, and K.-H. Yoo, "Improvement of Kafka streaming using partition and multi-threading in big data environment," *Sensors*, vol. 19, no. 1, p. 134, Jan. 2019.
- [6] X. Xu and Q. Hua, "Industrial big data analysis in smart factory: Current status and research strategies," *IEEE Access*, vol. 5, pp. 17543–17551, 2017.
- [7] P. Le Noac'h, A. Costan, and L. Bougé, "A performance evaluation of Apache Kafka in support of big data streaming applications," in *Proc. IEEE Int. Conf. Big Data*, Dec. 2017, pp. 4803–4806.
- [8] A. Bandi and J. A. Hurtado, "Big data streaming architecture for edge computing using Kafka and Rockset," in *Proc. 5th Int. Conf. Comput. Methodol. Commun. (ICCMC)*, Apr. 2021, pp. 323–329.
- [9] L. Lavanya, S. Venkatanarayanan, and A. A. Bhoraskar, "Real-time weather analytics: An end-to-end big data analytics service over Apache Spark with Kafka and long short-term memory networks," *Int. J. Web Services Res.*, vol. 17, no. 4, pp. 15–31, Oct. 2020.
- [10] Y. Zhang, T. Huang, and E. F. Bompard, "Big data analytics in smart grids: A review," *Energy Informat.*, vol. 1, no. 1, pp. 1–24, Dec. 2018.

- [11] B. Dhupia, M. U. Rani, and A. Alameen, "The role of big data analytics in smart grid management," in *Emerging Research in Data Engineering Systems and Computer Communications*. Singapore: Springer, 2020, pp. 402–412.
- [12] M. H. Ansari, V. T. Vakili, and B. Bahrak, "Evaluation of big data frameworks for analysis of smart grids," *J. Big Data*, vol. 6, no. 1, pp. 1–14, Dec. 2019.
- [13] R. Sahal, J. G. Breslin, and M. I. Ali, "Big data and stream processing platforms for Industry 4.0 requirements mapping for a predictive maintenance use case," *J. Manuf. Syst.*, vol. 54, pp. 138–151, Jan. 2020.
- [14] K. M. M. Thein, "Apache Kafka: Next generation distributed messaging system," *Int. J. Sci. Eng. Technol. Res.*, vol. 3, no. 47, pp. 9478–9483, 2014.
- [15] S. Vyas, R. K. Tyagi, C. Jain, and S. Sahu, "Performance evaluation of Apache Kafka—A modern platform for real time data streaming," in *Proc. 2nd Int. Conf. Innov. Practices Technol. Manage. (ICIPTM)*, vol. 2, Feb. 2022, pp. 465–470.
- [16] V.-A. Zamfir, M. Carabas, C. Carabas, and N. Tapus, "Systems monitoring and big data analysis using the elasticsearch system," in *Proc. 22nd Int. Conf. Control Syst. Comput. Sci. (CSCS)*, May 2019, pp. 188–193.
- [17] *Elastic Search: Introduction, Basics, Architecture and Usage of Elastic Search*. [Online]. Available: <https://hassantariqblog.wordpress.com/2016/09/23/elasticsearch-introduction-basics-architecture-and-usage-of-elastic-search/>
- [18] D. A. Osvik, J. W. Bos, D. Stefan, and D. Canright, "Fast software AES encryption," in *Proc. Int. Workshop Fast Softw. Encryption*. Berlin, Germany: Springer, 2010, pp. 75–93.
- [19] S. Heron, "Advanced Encryption Standard (AES)," *Netw. Secur.*, vol. 2009, no. 12, pp. 8–12, Dec. 2009.
- [20] C. Giblin, S. Rooney, P. Vetsch, and A. Preston, "Securing Kafka with encryption-at-rest," in *Proc. IEEE Int. Conf. Big Data (Big Data)*, Dec. 2021, pp. 5378–5387.



**SANGIL PARK** (Member, IEEE) received the B.E. degree from the Department of Computer Science Engineering, Chonnam National University, Republic of Korea, in February 2003, the M.E. degree from the Department of Software Engineering, Korea University, Seoul, Republic of Korea, in February 2017, and the Ph.D. degree from the Department of Data Informatics, Graduate School, National Korea Maritime and Ocean University, Busan, Republic of Korea, in February 2023.

He was the Department Manager of Samsung SDS, Samsung, Republic of Korea, from March 2002 to September 2008. He was a Research and Development Site Leader of the Next Generation Manufacturing Development Team, SK C&C, SK, Republic of Korea, from June 2011 to August 2022. He was an Adjunct Professor with the Department of Energy Convergence Security, Catholic University of Pusan, Republic of Korea, from September 2022 to August 2023. Since September 2022, he has been the CEO of MyLink Company Ltd. (Big Data Start-Up Company), Seoul.

Dr. Park received the 2019 Best Project Award (SK I Battery Charger/Discharger Battery Power Management System), SK holdings C&C, SK.



**JUN-HO HUH** (Member, IEEE) received the B.S. degree in science from the Department of Major of Applied Marine Sciences, the B.E. degree in engineering (double major) from the Department of Computer Engineering, Jeju National University at Ara, Jeju, Republic of Korea, in August 2007, the M.A. degree in education from the Department of Computer Science Education, Pukyong National University at Daeyeon, Busan, Republic of Korea, in August 2012, and the Ph.D. degree in engineering from the Department of Major of Computer Engineering, Graduate School, Pukyong National University at Daeyeon, Busan, Republic of Korea, in February 2016.

He finished the Cooperative Marine Science and Engineering Program, Texas A&M University at Galveston, Galveston, TX, USA, in August 2006. He was the General/Head Professor with the Catholic University of Pusan at the International Game Exhibition G-Star 2017 (G-Star 2017). He was a Research Professor with Dankook University, Jukjeon, Yongin, Republic of Korea, from July 2016 to September 2016. He was an Assistant Professor with the Department of Software, Catholic University of Pusan, Republic of Korea, from December 2016 to August 2019. He was an Assistant Professor with the Department of Data Informatics, National Korea Maritime and Ocean University, Busan, from September 2019 to September 2021. Since October 2021, he has been an Associate Professor (Tenured) with the Department of Data Informatics/Data Science, National Korea Maritime and Ocean University. Since September 2020, he has been the Center Chair (Director) of the Big Data Center for Total Lifecycle of Shipbuilding and Shipping, National Korea Maritime and Ocean University. Since September 2022, he has been a Join Associate Professor (Tenured) of the Global R&E Program for Interdisciplinary Technologies of Ocean Renewable Energy (BK 21 Four Research Group), Interdisciplinary Major of Ocean Renewable Energy Engineering, National Korea Maritime and Ocean University. He is the author of the book *Smart Grid Test Bed Using OPNET and Power Line Communication* (USA: IGI Global, 2017, pp.1-425) and *Principles, Policies, and Applications of Kotlin Programming* (USA: IGI Global, 2023, pp.1-457). He has authored/edited ten books and edited ten special issues in reputed Clarivate Analytics Index journals. He has also published more than 100 articles in Clarivate Analytics Index (SCI/SCIE/SSCI indexed) with over 2850 citations and has an H-index of 29.

Dr. Huh has been a Technical Committee (TC) Member of the International Federation of Automatic Control (IFAC), CC 1 (Systems and Signals), TC 1.5. (Networked Systems), since 2017. Since 2017, he has been a Technical Committee (TC) Member of IFAC, CC 3 (Computers, Cognition and Communication), and TC 3.2 (Computational Intelligence in Control). Since 2017, he has been a Technical Committee (TC) Member of IFAC, CC 7 (Transportation and Vehicle Systems), TC 7.2. (Marine Systems). Since 2020, he has been a Technical Committee (TC) Member of IFAC, CC 2 (Design Methods), TC 2.6. (Distributed Parameter Systems). He received the Best Paper Minister Award (Ministry of Trade, Industry and Energy, Korean Government) from the 16th International Conference on Control, Automation and Systems, in October 2016, ICROS with IEEE Xplore. He received the Springer Nature Most Cited Paper Award and the *Human-Centric Computing and Information Sciences* Most Cited Paper Award (research published in the journal during 2016–2018; SCIE IF=6.558), in 2019. He was the Organizing Chair of the 15th International Conference on Multimedia Information Technology and Applications (MITA 2019: University of Economics and Law (UEL), Vietnam National University Ho Chi Minh City, Vietnam). He was the Organizing Chair of the 17th International Conference on Multimedia Information Technology and Applications (MITA 2021: Jeju KAL Hotel, ROK). He was the Managing Editor (ME) of the *Journal of Information Processing Systems* (JIPS), Korea Information Processing Society (Scopus/ESCI indexed), from January 2020 to December 2021. He was the Managing Editor (ME) of the *Journal of Multimedia Information System* (JMIS), Korea Multimedia Society (EI/KCI indexed), from January 2017 to December 2022. He is also an Associate Editor (AE) of the *Journal of Information Processing Systems* (JIPS), Korea Information Processing Society (Scopus/ESCI indexed). He is also an Associate Editor (AE) of the *Journal of Multimedia Information System* (JMIS), Korea Multimedia Society (EI/KCI indexed). He is also an Associate Editor (AE) of *Human-Centric Computing and Information Sciences* (HCIS) (Springer Berlin Heidelberg (SCIE IF=6.558)). More information can be found at: <https://scholar.google.co.kr/citations?user=cr5wjNYAAAAJ&hl=en>.

•••