International Journal of Advanced Smart Convergence Vol.13 No.3 135-142 (2024)
http://dx.doi.org/10.7236/IJASC.2024.13.3.135

IJASC 24-3-15

Study on Proactive Data Process Orchestration in Distributed Cloud

Jong-Sub Lee*, Seok-Jae Moon**

**Professor, College of General Education, Semyung University, Jecheon, Korea*
***Professor, Department of Artificial Intelligence Institute of Information Technology, KwangWoon University, Korea*
E-mail: 99jslee@semyung.ac.kr, msj8086@kw.ac.kr

Abstract

Recently, along with digital transformation, technologies such as cloud computing, big data, and artificial intelligence have been actively introduced. In a situation where these technological changes are progressing rapidly, it is often difficult to manage processes efficiently using existing simple workflow management methods. Companies providing current cloud services are adopting virtualization technologies, including virtual machines (VMs) and containers, in their distributed system infrastructure for automated application deployment. Accordingly, this paper proposes a process-based orchestration system for integrated execution of corporate process-oriented workloads by integrating the potential of big data and machine learning technologies. This system consists of four layers as components for performing workload processes. Additionally, a common information model is applied to the data to efficiently integrate and manage the various formats and uses of data generated during the process creation stage. Moreover, a standard metadata protocol is introduced to ensure smooth exchange between data. This proposed system utilizes various types of data storage to store process data, metadata, and analysis models. This enables flexible management and efficient processing of data.

Keywords: *: Distributed Cloud, Data Process, Cloud Computing, Orchestration, Machine Learning, EMRA*

1. INTRODUCTION

Recently, cloud-based platform technology has been actively introduced [1]. With technological changes progressing rapidly, companies often find it difficult to efficiently manage business processes using traditional simple workflow management methods [2]. In this paper, we propose an orchestration system for proactive data processing in a distributed cloud for workload execution. The proposed system consists of four layers: business process-based query layer, process layer, XMDR [3] layer (Information Layer), and resource layer. This hierarchical configuration applies a common information model to data to efficiently integrate and manage the various formats and uses of data that occur during the workload process creation stage. Additionally, the EMRA [4] protocol, a standard metadata, is introduced for smooth interoperability between

Manuscript Received: July. 16. 2024 / Revised: July. 22. 2024 / Accepted: July. 28. 2024
Corresponding Author: msj8086@kw.ac.kr
Tel: 02-940-8283, Fax: 02-940-5443
Author's affiliation: Professor, Graduate School of Smart Convergence, KwangWoon University, Seoul, Korea

data. The system utilizes various types of data storage to store in-process data, metadata, and analytical models, allowing flexible management and efficient processing in data interoperability. Additionally, focusing on big data generated in the industrial automation field, the paper presents the procedures necessary for recognizing hidden patterns and building predictive models using machine learning. In particular, when considering machine learning use cases, training data is first loaded into a data store. This data goes through a pre-processing process for model learning and then proceeds to the optimal model selection and application stage. The selected model is evaluated using test data, and this entire process is executed within a batch processing framework. Additionally, a visual layer is provided to intuitively express the analyzed prediction results, making the results more accessible and actionable for users. Chapter 2 describes related research, and Chapter 3 details the components and operation scenarios of the proposed system. Chapter 4 outlines the experimental test environment and presents the experimental results. Finally, Chapter 5 concludes with a discussion of the findings and suggestions for future work.

2. RELATED WORK

The benefits of PCA (Principal Component Analysis) [5] data analysis architecture in a corporate environment are manifold. In particular, it allows learned models to be quickly and accurately applied to current process data. Using data analysis techniques such as PCA, high-dimensional data can be reduced to low dimensions, reducing the complexity of analysis and increasing computational efficiency. This approach plays a significant role in versatile enterprise applications such as condition monitoring, anomaly detection, production process optimization, and customer behavior analysis. In a cloud computing environment, a workload [6] consists of a combination of multiple IT assets, which may include virtual machines (VMs), databases, applications, APIs, network resources, and storage. These assets support one or more business processes or tasks and may interact with each other and/or depend on other IT assets or a larger platform. The components of Extended Metadata Registry Access (EMRA) include a central metadata registry, extension modules, API gateway, access control system, and monitoring and logging system [7]. Through this, metadata is collected and stored from various data sources, access is allowed or restricted through API according to user permissions, and monitoring and logging are performed in real time to ensure scalability and security at the same time.

3. PROPOSED SYSTEM

3.1. Component

We propose an orchestration system for process-based queries that execute workloads in an integrated manner. As an application of the Lambda architecture, this proposed system can efficiently perform large-scale batch data processing in addition to real-time processing of stream data. This makes it possible to flexibly respond to various workloads and data types that occur within the process, allowing end users to make faster and more accurate business decisions. To further enhance this flexibility and scalability, the main contribution proposed in this paper is to efficiently manage and utilize various computing resources through orchestration technology. The proposed system consists of different layers as shown in Figure 1.

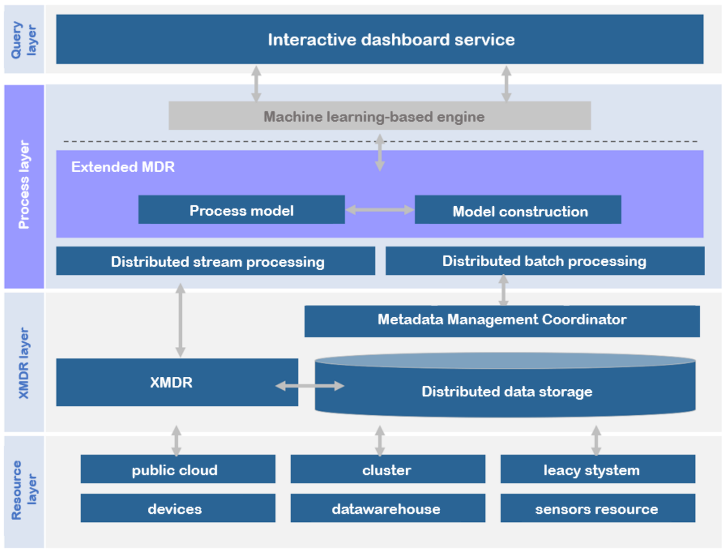


Figure 1. The Conceptual Architecture

As an application of the Lambda architecture, this proposed system can efficiently perform large-scale batch data processing in addition to real-time processing of stream data. This makes it possible to flexibly respond to various workloads and data types that occur within the process, allowing end users to make faster and more accurate business decisions. To further enhance this flexibility and scalability, the main contribution proposed in this paper is to efficiently manage and utilize various computing resources through orchestration technology. The proposed system consists of different layers as shown in Figure 1.

- **Query layer.** This layer specifies business needs, use cases, and tasks to be handled. In particular, the connectivity of the system for data sharing must be expanded and visible. Visual interfaces and dashboards are therefore important for understanding difficult concepts or identifying hidden patterns within process data. The visualization layer displays prediction results and allows adding expert knowledge in the form of semantic annotations to facilitate data analysis tasks. Therefore, this layer includes interactive analytics software, dashboards, and client applications.

- **Process layer.** This layer analyzes process internal data to identify hidden patterns and builds an ML prediction model. Training data for each machine learning use case is loaded from the data store and preprocessed for model training. The next step is to select and apply an appropriate model using the training data. Afterwards, the model is evaluated using the training data. These steps are called model construction and can be performed in distributed batch processing. Once the trained model is suitable to solve the query problem, it can be deployed into distributed stream processing for online prediction on stream data.

- **Metadata layer.** At this layer, it is applied in a metadata registry or a standardized information model of a specific branch suitable for data interoperability in the process. Metadata is maintained at this layer for data access and analysis. Accordingly, this paper applies EMRA [8], an extended version of metadata. Additionally, various types of distributed data storage are used to store process data, metadata analysis, and models.

- **Resource layer.** This layer includes devices, sensors, clusters, DW, legacy systems, etc. These elements constitute a company's core data source and are converted into various forms of data. Since this data has various formats and uses, it needs to be integrated through a common information model using a public cloud or cluster. This encompasses the comprehensive process of data collection, integration, and the transition between the physical and digital worlds.

3.2. System Operation flowchart

The operation scenario of the system proposed in this paper is shown in Figure 2.

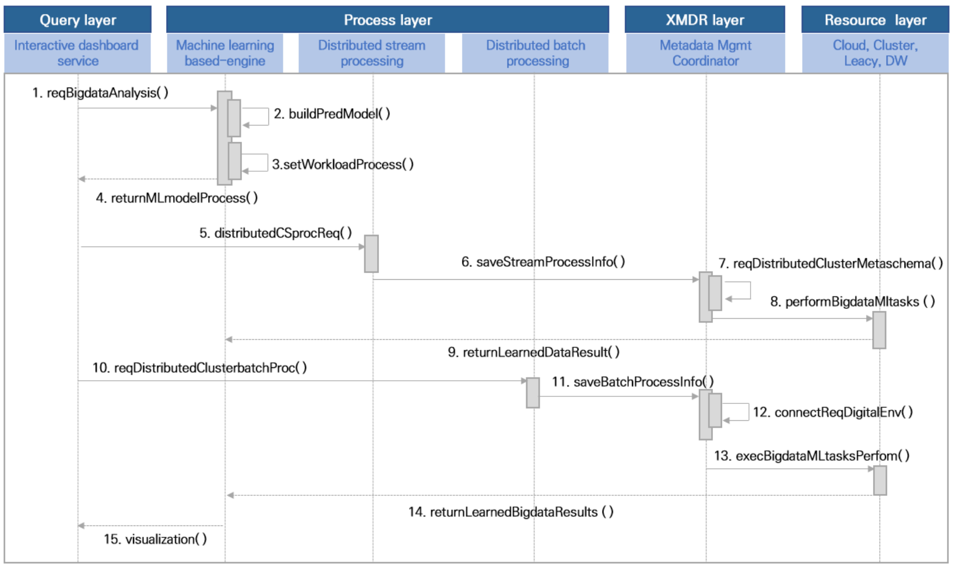

Query layer
Process layer
XMDR layer
Resource layer
Interactive dashboard service
Machine learning based-engine
Distributed stream processing
Distributed batch processing
Metadata Mgmt Coordinator
Cloud, Cluster, Leacy, DW
1. reqBigdataAnalysis( )
2. buildPredModel( )
3.setWorkloadProcess( )
4. returnMLmodelProcess( )
5. distributedCSprocReq( )
6. saveStreamProcessInfo( )
7. reqDistributedClusterMetaschema( )
8. performBigdataMltasks ( )
9. returnLearnedDataResult( )
10. reqDistributedClusterbatchProc( )
11. saveBatchProcessInfo( )
12. connectReqDigitalEnv( )
13. execBigdataMLtasksPerfom( )
14. returnLearnedBigdataResults ( )
15. visualization( )

Figure 2. The System Flowchart

1. When you access the system dashboard, you can immediately and visually check the current status of your workload. This includes real-time updates on inventory levels, machine status, and throughput throughout the entire production process. With real-time monitoring all data generated by the machine, from operational statistics to error notifications, is displayed immediately. This allows us to respond quickly to urgent issues, minimizing downtime. For example, if the machine failure rate predicted by the machine learning model exceeds a certain threshold, an immediate warning is issued to enable proactive response.

2. Data analysis algorithms process the incoming data stream to detect signs or patterns of machine performance degradation. Using real data analysis and machine learning models, we predict the optimal maintenance schedule for machines showing signs of poor performance or failure. For example, by analyzing data such as temperature, pressure, and humidity collected from sensors, we predict the possibility of machine failure or develop a model that automatically classifies product quality. This predictive maintenance approach significantly reduces unplanned downtime by proactively addressing potential defects. Based on this predicted data, managers plan maintenance schedules and issue work orders to ensure smooth operation.

3. All data in the product line is stored via EMRA along with metadata. All data generated or collected in the smart factory is centrally managed. This layer stores sensor data, operational records, metadata, etc. This layer maintains data consistency and integrity, and performs indexing and tagging to easily extract data for necessary analysis or prediction. The architecture is designed to enable stable storage and retrieval of data even in a highly distributed environment. This structured approach facilitates easy retrieval and makes accumulated

data immediately available for future analysis and reporting. In other words, beyond simple storage, this layer provides advanced data analysis capabilities to gain real insights from accumulated data.

4. It plays a role in connecting physical assets, machines, sensors, and inventory within the factory with the digital environment. The system integrates various data formats and protocols to create a single integrated data stream. Since a smart factory is a dynamic environment with the possibility of introducing new machines or upgrading existing machines, the integration of these new assets is important. For example, data collected from IoT sensors is preprocessed here and, if necessary, sent to a data store in the information layer or a machine learning model in the functional layer. Through this process, the physical and digital worlds are seamlessly connected, increasing the efficiency and level of automation in smart factories.

5. Smart factory workloads are continuously monitored to maintain optimal work efficiency. The system's responsiveness allows for quick readjustments, ensuring maximum operational efficiency. With the support of a container orchestration system, all resources (from human labor to machines) within the smart factory are fully utilized to achieve optimal production results.

4. Experimental analysis and results

This paper proposes a system tested across several virtual machines to construct the necessary cluster for implementing PCA (Principal Component Analysis) parallel computing in a big data environment. To implement parallel computation of PCA, we created the necessary clusters by creating several virtual machines in the big data cluster. The virtual machine resource specifications applied are detailed in Table 1. Information about the technology, infrastructure, and configuration required for the experiment is shown in Table 2.

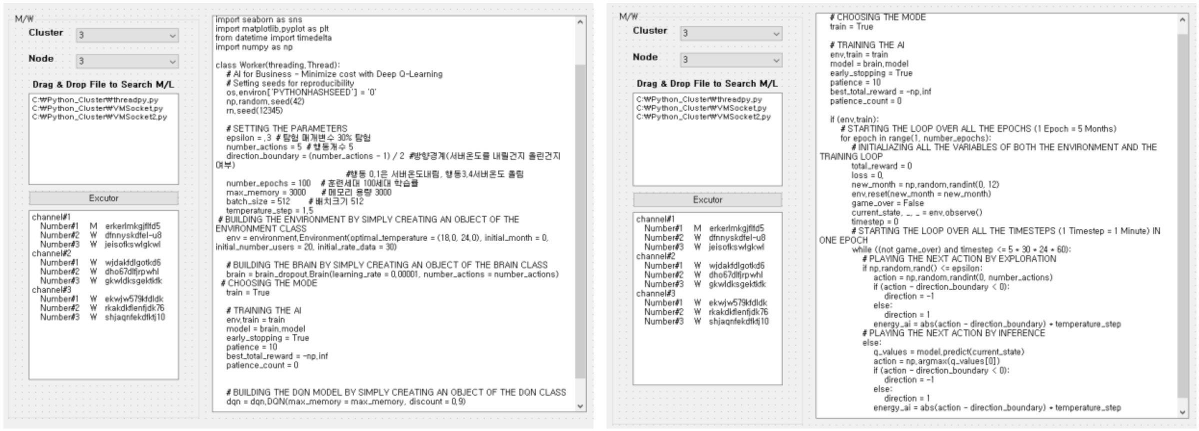
Table 1. VM Resource Specification

Item	Details
Processor	- Intel(R) Core(TM) i5-4670 CPU @ 3.40GHz 3.40 GHz
	- Virtual CPUs (4 sockets with 2 cores per socket)32
Memory/Storage	16GB / 700GB SSD
	1 Gbit/s network card
Network	Ubuntu 16.04 xenial
OS	

Table 2. Cluster Configuration Specification

Category	Framework	Description
Data Ingestion	Kafka Cluster	3 brokers, 20 partitions for the input and output topics
Data Storage	Hadoop Cluster	1 name node and 3 data nodes
	MongoDB	single node
	Influx DB	single node
Batch Processing	Spark Cluster	1 master node and 3 workers
Stream Processing	Kubernetes Cluster	1 master and 3 nodes

The specifications for the proposed system experiment ensure diverse and integrated data processing capabilities, high scalability, data stability, and flexibility. Real-time and batch data processing is possible, and the cluster structure can flexibly respond to increases in data volume. The experiment enables users to easily and efficiently work with data through an integrated interface, enhancing the architecture's reliability through experimental verification. Figure 3 implements approximately 36 million historical measurement vectors (each with 41 components) consisting of air conditioning management process data into a big data platform as a data set.



```
Cluster 3
Node 3

C:₩Python_Cluster₩threadpy.py
C:₩Python_Cluster₩VMSocket.py
C:₩Python_Cluster₩VMSocket2.py

channel#1
   Number#1   M   erkerlmkgjflfd5
   Number#2   W   dfnnyskdfel-u8
   Number#3   W   jeisofkswlgkwl
channel#2
   Number#1   W   wjdakfdlgotkd6
   Number#2   W   dho67dlfjrpwhl
   Number#3   W   gkwldksgektkfk
channel#3
   Number#1   W   ekwjw579kfdldk
   Number#2   W   rkakdkflenfjdk76
   Number#3   W   shjaqnfekdfktj10
```

```
import seaborn as sns
import matplotlib.pyplot as plt
from datetime import timedelta
import numpy as np

class Worker(threading.Thread):
    # AI for Business - Minimize cost with Deep Q-Learning
    # Setting seeds for reproducibility
    os.environ['PYTHONHASHSEED'] = '0'
    np.random.seed(42)
    rn.seed(12345)

    # SETTING THE PARAMETERS
    epsilon = .3  # 탐험 매개변수 30% 탐험
    number_actions = 5  # 행동개수 5
    direction_boundary = (number_actions - 1) / 2  #방향경계(서버온도를 내릴건지 올린건지
여부)
                                                  #행동 0,1은 서버온도내림, 행동3,4서버온도 올림
    number_epochs = 100    # 훈련세대 100세대 학습률
    max_memory = 3000      # 메모리 용량 3000
    batch_size = 512       # 배치크기 512
    temperature_step = 1.5
 # BUILDING THE ENVIRONMENT BY SIMPLY CREATING AN OBJECT OF THE
ENVIRONMENT CLASS
    env = environment.Environment(optimal_temperature = (18.0, 24.0), initial_month = 0,
initial_number_users = 20, initial_rate_data = 30)

    # BUILDING THE BRAIN BY SIMPLY CREATING AN OBJECT OF THE BRAIN CLASS
    brain = brain_dropout.Brain(learning_rate = 0.00001, number_actions = number_actions)
  # CHOOSING THE MODE
    train = True

    # TRAINING THE AI
    env.train = train
    model = brain.model
    early_stopping = True
    patience = 10
    best_total_reward = -np.inf
    patience_count = 0


    # BUILDING THE DQN MODEL BY SIMPLY CREATING AN OBJECT OF THE DQN CLASS
    dqn = dqn.DQN(max_memory = max_memory, discount = 0.9)
```

```
Cluster 3
Node 3

C:₩Python_Cluster₩threadpy.py
C:₩Python_Cluster₩VMSocket.py
C:₩Python_Cluster₩VMSocket2.py

channel#1
   Number#1   M   erkerlmkgjflfd5
   Number#2   W   dfnnyskdfel-u8
   Number#3   W   jeisofkswlgkwl
channel#2
   Number#1   W   wjdakfdlgotkd6
   Number#2   W   dho67dlfjrpwhl
   Number#3   W   gkwldksgektkfk
channel#3
   Number#1   W   ekwjw579kfdldk
   Number#2   W   rkakdkflenfjdk76
   Number#3   W   shjaqnfekdfktj10
```

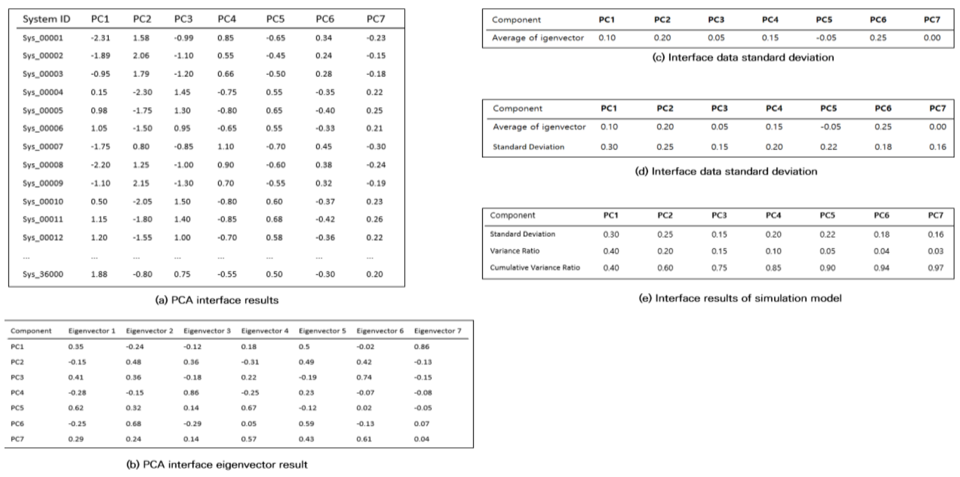
```
    # CHOOSING THE MODE
    train = True

    # TRAINING THE AI
    env.train = train
    model = brain.model
    early_stopping = True
    patience = 10
    best_total_reward = -np.inf
    patience_count = 0

    if (env.train):
        # STARTING THE LOOP OVER ALL THE EPOCHS (1 Epoch = 5 Months)
        for epoch in range(1, number_epochs):
            # INITIALIAZING ALL THE VARIABLES OF BOTH THE ENVIRONMENT AND THE
TRAINING LOOP
            total_reward = 0
            loss = 0.
            new_month = np.random.randint(0, 12)
            env.reset(new_month = new_month)
            game_over = False
            current_state, _, _ = env.observe()
            timestep = 0
            # STARTING THE LOOP OVER ALL THE TIMESTEPS (1 Timestep = 1 Minute) IN
ONE EPOCH
            while ((not game_over) and timestep <= 5 * 30 * 24 * 60):
                # PLAYING THE NEXT ACTION BY EXPLORATION
                if np.random.rand() <= epsilon:
                    action = np.random.randint(0, number_actions)
                    if (action - direction_boundary < 0):
                        direction = -1
                    else:
                        direction = 1
                    energy_ai = abs(action - direction_boundary) * temperature_step
                # PLAYING THE NEXT ACTION BY INFERENCE
                else:
                    q_values = model.predict(current_state)
                    action = np.argmax(q_values[0])
                    if (action - direction_boundary < 0):
                        direction = -1
                    else:
                        direction = 1
                    energy_ai = abs(action - direction_boundary) * temperature_step
```

Figure 3. Test Simulation Model Interface

The simulation model interface attempts to perform the task of training a deep Q-learning model in parallel by leveraging Python's threading library. In particular, the Worker class inherits threading. Thread and is designed to create multiple threads and run them in parallel. Each thread is responsible for setting up an environment, constructing a deep learning model, and using this model to learn optimal behavior in a given environment. In the part where hyperparameters are set, the exploration rate, number of actions, number of epochs, etc. are defined, and standards for how the AI should behave under specific conditions are established. Training sessions alternate between exploration and exploitation, updating the state of the environment and attempting to optimize the AI's rewards. This process is repeated until a certain condition is met or an early stopping condition is reached. Additionally, the AI's performance can be tracked for each epoch, and learning can be stopped early through a patience mechanism if there is no improvement. Ultimately, each thread is responsible for independently training AI models and storing the models with the best performance.



System ID	PC1	PC2	PC3	PC4	PC5	PC6	PC7
Sys_00001	-2.31	1.58	-0.99	0.85	-0.65	0.34	-0.23
Sys_00002	-1.89	2.06	-1.10	0.55	-0.45	0.24	-0.15
Sys_00003	-0.95	1.79	-1.20	0.66	-0.50	0.28	-0.18
Sys_00004	0.15	-2.30	1.45	-0.75	0.55	-0.35	0.22
Sys_00005	0.98	-1.75	1.30	-0.80	0.65	-0.40	0.25
Sys_00006	1.05	-1.50	0.95	-0.65	0.55	-0.33	0.21
Sys_00007	-1.75	0.80	-0.85	1.10	-0.70	0.45	-0.30
Sys_00008	-2.20	1.25	-1.00	0.90	-0.60	0.38	-0.24
Sys_00009	-1.10	2.15	-1.30	0.70	-0.55	0.32	-0.19
Sys_00010	0.50	-2.05	1.50	-0.80	0.60	-0.37	0.23
Sys_00011	1.15	-1.80	1.40	-0.85	0.68	-0.42	0.26
Sys_00012	1.20	-1.55	1.00	-0.70	0.58	-0.36	0.22
...	...	...	...	...	...	...	...
Sys_36000	1.88	-0.80	0.75	-0.55	0.50	-0.30	0.20

(a) PCA interface results

Component	Eigenvector 1	Eigenvector 2	Eigenvector 3	Eigenvector 4	Eigenvector 5	Eigenvector 6	Eigenvector 7
PC1	0.35	-0.24	-0.12	0.18	0.5	-0.02	0.86
PC2	-0.15	0.48	0.36	-0.31	0.49	0.42	-0.13
PC3	0.41	0.36	-0.18	0.22	-0.19	0.74	-0.15
PC4	-0.28	-0.15	0.86	-0.25	0.23	-0.07	-0.08
PC5	0.62	0.32	0.14	0.67	-0.12	0.02	-0.05
PC6	-0.25	0.68	-0.29	0.05	0.59	-0.13	0.07
PC7	0.29	0.24	0.14	0.57	0.43	0.61	0.04

(b) PCA interface eigenvector result

Component	PC1	PC2	PC3	PC4	PC5	PC6	PC7
Average of igenvector	0.10	0.20	0.05	0.15	-0.05	0.25	0.00

(c) Interface data standard deviation

Component	PC1	PC2	PC3	PC4	PC5	PC6	PC7
Average of igenvector	0.10	0.20	0.05	0.15	-0.05	0.25	0.00
Standard Deviation	0.30	0.25	0.15	0.20	0.22	0.18	0.16

(d) Interface data standard deviation

Component	PC1	PC2	PC3	PC4	PC5	PC6	PC7
Standard Deviation	0.30	0.25	0.15	0.20	0.22	0.18	0.16
Variance Ratio	0.40	0.20	0.15	0.10	0.05	0.04	0.03
Cumulative Variance Ratio	0.40	0.60	0.75	0.85	0.90	0.94	0.97

(e) Interface results of simulation model

Figure 4. Test simulation result: (a) PCA interface results, (b) PCA interface eigenvector result, (c) Interface data standard deviation, (d) Interface data standard deviation, (e) Interface results of simulation model

Training sessions alternate between exploration and exploitation, updating the state of the environment and attempting to optimize the AI's rewards. This process is repeated until a certain condition is met or an early stopping condition is reached. Additionally, the AI's performance can be tracked for each epoch, allowing for early termination of learning through a patience mechanism in the absence of improvement. Ultimately, each thread is responsible for independently training AI models and storing the models with the best performance. The main thread creates and executes these worker threads, terminating upon completion of all tasks. <Figure 4> (a) shows the model interface results. The results obtained through PCA list the names of several entities and their corresponding PCA scores. We are currently performing principal component analysis (PCA) on heating and cooling system data, which is effectively reducing the dimensionality of the original data. Through this analysis, a complex dataset originally consisting of 41 variables is condensed into a few highly volatile key elements. This process greatly improves computational efficiency and provides information in a refined form that is less sensitive to noise in the data. <Figure 4> (b) is the interface eigenvector result. Here, each given row represents a principal component, and each column is an element of the corresponding eigenvector. These elements represent the weight of the eigenvector for each feature (41 components) of the original dataset. For example, the Eigenvector 1 column defines the direction of PC1, which represents the greatest variability in the data. Different eigenvectors represent variability in different directions in the dataset. Eigenvectors vary depending on the structure and pattern of the data. Each principal component direction is a new axis in the original data space, and the eigenvector elements quantify the contribution of the original features to this new axis. Figure 4 (c) Average of interface data. The data average represents the average value of each element of the eigenvector. The 'Average of Eigenvector' column shows the average values for each element across principal components (PC1, PC2, ..., PC7). These average values are the average calculated for each dimension of the vector representing the direction of each principal component. <Figure 4> (d) Standard deviation of interface data. It represents the standard deviation for various variables derived through PCA analysis. The displayed values reflect the volatility and spread of the data distribution for each variable, and a higher standard deviation means that the data values of that variable are more spread out from the mean. These standard deviation values play an important role in interpreting the results of PCA analysis or evaluating the stability and consistency of data. Additionally, it can be useful in determining the degree of dispersion of observed values for each variable. Figure 4 (d) is the interface result of the simulation model. The standard deviation of each principal component represents the contribution of that principal component to the variance of the entire dataset. The ratio of variance is the square of the standard deviation of the corresponding principal component divided by the total variance. The ratio of cumulative variance represents the cumulative sum of the variance ratios. Information obtained through principal component analysis (PCA) results demonstrates the effectiveness of PCA in reducing data dimensionality and summarizing information. In particular, it is revealed that the first principal component (PC1) contains the most information, accounting for 40% of the data variability. This suggests that PC1, as the main axis capturing the most variation, contains significant information, offering a clearer understanding of the dataset's intrinsic structure and patterns. The cumulative variance ratio of the top three principal components (PC1, PC2, PC3) containing the main information of the dataset reaches 75%, confirming that they represent important characteristics of the data. These results mean that much of the important information in the original dataset can be maintained even when the data is reduced.

5. CONCLUSION

This paper proposes a system that efficiently integrates complex processes within a company in real time in a distributed cloud. This system facilitates data integration with a metadata-centric approach, aiming to integrate various data sources and types in the cloud, effectively enabling data exchange. Through experiments, this paper demonstrated that the proposed system processed a large-scale air conditioning management data set, offering faster processing speed and superior scalability compared to traditional batch processing methods. The system proposed in this paper enables data integration by sharing the same information model, thus facilitating smooth data exchange among heterogeneous systems. Additionally, the system offers scalability across various layers, along with redundancy, enhancing its capability for parallel PCA implementation. In conclusion, the research results of this system provide an important reference point for developing new architectures and methodologies for data-driven business process management and optimization. This is expected to have a significant impact on the future development direction of cloud computing, big data, and machine learning technologies.

ACKNOWLEDGMENT

This paper was supported by the Semyung University Research Grant of 2024.

REFERENCES

[1] Thavi, Riddhi, et al. "Role of cloud computing technology in the education sector." Journal of Engineering, Design and Technology 22.1, pp.182-213, 2024.
DOI: https://doi.org/10.1108/JEDT-08-2021-0417

[2] Broccardo, Laura, et al. "Business processes management as a tool to enhance intellectual capital in the digitalization era: the new challenges to face." Journal of Intellectual Capital 25.1, pp.60-91, 2024.
DOI: https://doi.org/10.1108/JIC-04-2023-0070

[3] Lee, Jong-Sub, and Seok-Jae Moon. "Business Collaborative System Based on Social Network Using MOXMDR-DAI+." International Journal of Advanced Culture Technology 8.3, pp.223-230, 2020.
DOI: https://doi.org/10.17703/IJACT.2020.8.3.223

[4] He, Xindong. "Principal Component Analysis (PCA)." Geographic Data Analysis Using R. Singapore: Springer Nature Singapore, pp.155-165, 2024.
DOI: https://doi.org/10.1007/978-981-97-4022-2_8

[5] Khan, Habib Ullah, Farhad Ali, and Shah Nazir. "Systematic analysis of software development in cloud computing perceptions." Journal of Software: Evolution and Process 36.2, 2024.
DOI: https://doi.org/10.1002/smr.2485

[6] Dakić, Vedran, Mario Kovač, and Jurica Slovinac. "Evolving High-Performance Computing Data Centers with Kubernetes, Performance Analysis, and Dynamic Workload Placement Based on Machine Learning Scheduling." Electronics 13.13, 2651, 2024.
DOI: https://doi.org/10.3390/electronics13132651

[7] Park, Ho-Kyun, and Seok-Jae Moon. "Distributed Data Platform Collaboration Agent Design Using EMRA." International Journal of Internet, Broadcasting and Communication 14.2, pp.40-46, 2020.
DOI: https://doi.org/10.7236/IJIBC.2022.14.2.40

[8] Manivannan, R., et al. "Performance enhancement of cloud security with migration algorithm for choosing virtual machines in cloud computing." Engineering Research Express 6.1, 015204, 2024.
DOI: https://doi.org/10.1088/2631-8695/ad2ef9