Copyright ⓒ 2023 KSII

Resource Metric Refining Module for AIOps Learning Data in Kubernetes Microservice

Jonghwan Park, Jaegi Son, and Dongmin Kim*
Korea Electronics Technology Institute
Seongnam, Gyeonggi-do, South Korea
[e-mail: bomebug15@gmail.com, jgson@keti.re.kr, dmkim@keti.re.kr]
*Corresponding author: Dongmin Kim

Received August 17, 2022; revised January 10, 2023; revised March 23, 2023; revised May 3, 2023; accepted June 5, 2023; published June 30, 2023

Abstract

In the cloud environment, microservices are implemented through Kubernetes, and these services can be expanded or reduced through the autoscaling function under Kubernetes, depending on the service request or resource usage. However, the increase in the number of nodes or distributed microservices in Kubernetes and the unpredictable autoscaling function make it very difficult for system administrators to conduct operations. Artificial Intelligence for IT Operations (AIOps) supports resource management for cloud services through AI and has attracted attention as a solution to these problems. For example, after the AI model learns the metric or log data collected in the microservice units, failures can be inferred by predicting the resources in future data. However, it is difficult to construct data sets for generating learning models because many microservices used for autoscaling generate different metrics or logs in the same timestamp. In this study, we propose a cloud data refining module and structure that collects metric or log data in a microservice environment implemented by Kubernetes; and arranges it into computing resources corresponding to each service so that AI models can learn and analogize service-specific failures. We obtained Kubernetes-based AIOps learning data through this module, and after learning the built dataset through the AI model, we verified the prediction result through the differences between the obtained and actual data.

Keywords: Kubernetes, Cloud Data Preprocessing, Artificial Intelligence for IT operations, Resource Prediction, Microservice

A preliminary version of this paper was presented at APIC-IST 2022, and was selected as an outstanding paper**.**

http://doi.org/10.3837/tiis.2023.06.003 **ISSN : 1976-7277**

1. Introduction

A microservice refers to an independent service deployment method that does not affect the relationship between services, unlike the existing monolithic system that has dependency between the services. Regarding monolithic architecture, all services should be updated if the service traffic is excessively increased, but microservices can directly modify the service that has a failure [1]. Recently, the adoption of cloud-based microservice methods in service distribution methods has been increasing [2-6]. Kubernetes is a representative tool used to build a cloud, and using it, it is possible to easily expand or reduce service-specific resources by scaling them up according to service requests. If traffic increases in a particular microservice in Kubernetes, the service will be implemented within the range wherein the services are expanded or reduced according to their usage. However, the usage of all the services is not increased or decreased equally, and the manager needs to monitor and manage the characteristics of each service directly.

As a response to these problems, the number of research cases under Artificial Intelligence for IT Operations (AIOps) is increasing. AIOps collects cloud and infrastructure data, analyzes and patterns them through AI models, and then predicts the future usage and failures of the data [7,8]. However, there is no reference point for a separate data purification method for AIOps. Although there are studies on AIOps-related usage prediction, there are many on using it to learn benchmark dataset such as Google computing cluster and Backblaze disk statistics data sets, and few using custom data [9,19,20-24]. The dataset is somewhat different from the one collected directly from Kubernetes, and there is a disadvantage that it is difficult to apply it to actual services. To apply AI predictions of usage to an actual service, a system is needed that allows cloud engineers and data engineers to collaborate and extract and preprocess the data themselves. With the increasing number of large-scale services that offer more runtime performance and include more information for monitoring, it seems that there will be a growing need for microservice metric/log analysis and management, and a dataset suitable for microservice analysis using AIOps needs to be developed [28].

Cloud data have a characteristic time series that is recorded in real time and contains a huge amount of data because it records both generated service and container data. For example, when large-scale micro-services are deployed, the cloud records the service-specific metric/Log data. Due to the cloud characteristics of preventing application overload and stabilizing resources between services, a single task is performed on several nodes. Additionally, regarding Kubernetes, if the same service is placed on another node, several Pods will be created according to the hash value after the service name. There is simultaneous duplication of data even if it is the same service, and as abnormal time series data are generated, there is a problem in securing the data required for the AI/machine learning (ML) model learning. Additionally, there is no separate guideline for refining data, making data difficult to preprocess. **Fig. 1** shows an example of the process.

In this study, we propose a cloud data refining module for collecting data in a microservice environment under Kubernetes. The data refining module is designed to refine data for learning AI/ML models that can predict and infer failures by service. In Kubernetes through data refining module, after Metric / Log data is collected, the duplication value is pre-processed at each service and it refines to data capable of the AI / ML model is the learning. The data refining module was designed to collect data from Prometheus and perform time-to-time data matching and quarterly preprocessing for each service. Through the proposed module, cloud engineers and data engineers can easily generate AIOPS learning data without complicated processes, and obtain data that meets the characteristics of AIOps analysis at various levels such as Pod, Node, and Cluster. Due to the distributed nature of microservices, we collected function-specific metric average data for predicting resource usage. By providing average values instead of individual predictions for each service, we can ensure the diversity of analysis targets.

We obtained AIOps learning data suitable for analyzing resource usage in microservices under Kubernetes through the cloud data refining module, and validated it using the Informer model, which is suitable for long-term prediction of microservices. After learning the resource prediction process for each service through the Informer model, we verified the prediction results using the MSE and MAE indicators and confirmed that we can create a learnable dataset based on this. The proposed module allows for generating AI/ML models for predicting and inferring failures by service without the need for collaboration between cloud engineers and data engineers.

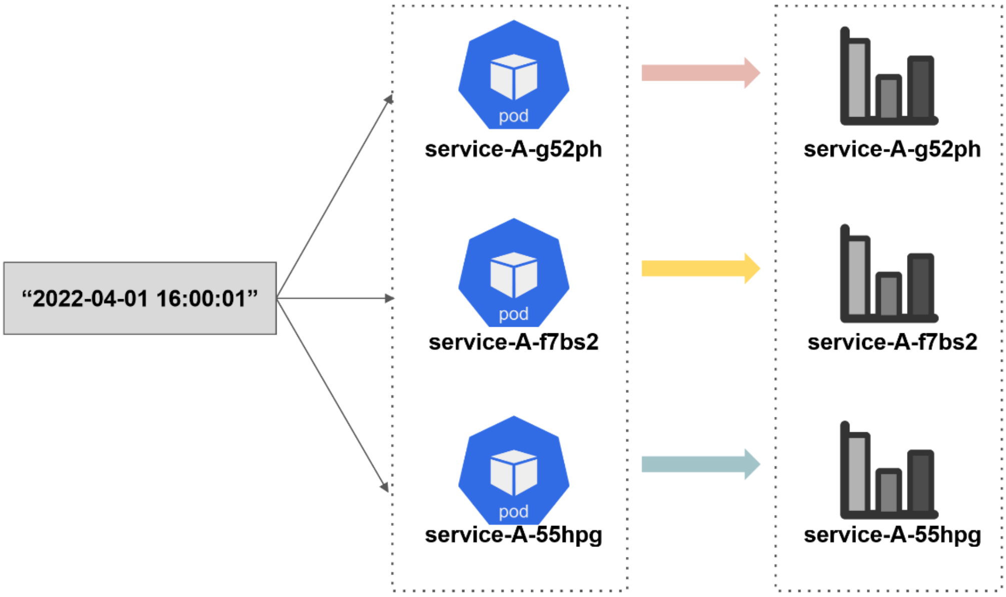


Fig. 1. Mismatch on same time-to-same service

2. Related Works

2.1 Cloud Metric/Log Data Application

There are studies on monitoring systems that use Prometheus & the Elasticsearch, Logstash, and Kibana (ELK) stack to collect and utilize data generated in the cloud. There are many studies [13-15] wherein Prometheus and Grafana are used to collect and monitor metric data generated in the system [6,10-12,20,21]; and Elasticsearch and Kibana are used to collect log data generated in the network. However, in most studies, these tools only monitored the data; there is only one research case [24] that directly collects and processes microservice data for AI/machine learning training.

2.2 Resource Prediction

For predicting cloud system utilization, we focused on ML/deep learning with the goal of predicting the workload, central processing unit (CPU), and random access memory (RAM) utilization. There were studies wherein the autoregressive integrated moving average (ARIMA) model, which is based on statistical techniques, was used [16,17]; the characteristics between cloud virtual machines (VMs) were classified through co-clustering, and the changes in workload patterns were predicted through the Hidden Markov model [18]. In this study, data on 21 days of CPU usage were used as learning data to observe the variations in the VMs. But these time-series data mining techniques have limitation in accuracy when applied in cloud environments due to their poor feature extraction capabilities. [19]

There have also been studies wherein predictions have been attempted using long short-term memory (LSTM) in Google Cluster Data and Unix Distributed System Load Tracking Data Set [19]. This was the first study to apply an RNN-based deep learning methodology for predicting host load, a metric used to measure system load. While the study demonstrated higher accuracy than data mining-based models, LSTM's limitations include its short prediction horizon and difficulty in parallel computation, making it unable to process large amounts of data. Another study utilized the ML-based Hybrid GA-PSO algorithm for multi-resource prediction [22]. By combining GA (Genetic Algorithm) to find global optima and PSO (Particle Swarm Optimization) to analyze possible solutions and find the best optimal value, the efficiency of the deep learning network's prediction was increased. The data was collected in Kubernetes, and the algorithm's accuracy was verified using the Google cluster dataset. Another study [23] proposed an intelligent resource management model based on VM failure, utilizing the Google cluster dataset as the training data for the FR-IRM ensemble-based model. The FR-IRM algorithm reduced active server and energy consumption by up to 51.2%.

A study [21] utilizing a commercial network operator dataset proposed a deep learning model to predict VNF (Virtual Network Function) instances in Kubernetes. The study predicted the variation of VNF instance through the model's learning results and applied them to simulate fluid cluster management. The FFNN, LSTM, and CNN-LSTM models were used for training, and the MSE, MAE, and RSME indicators were utilized for quantitative evaluation of the learning results. In other study [20] compared the CPU traffic prediction performance of LSTM, GRU, and 1D-CNN models using a Kubernetes testbed dataset. A module was created to generate REST API call patterns based on the Telecom Italia open-source dataset for 5G network traffic pattern analysis and prediction, creating a dataset that includes CPU changes. Another study [24] collected data using Prometheus extensions for microservices-based applications such as Cortex and optimized them using linear programming to predict microservice resource consumption and CPU optimization. However, the data pre-processing process for the creation of multiple pods from a single service, which is a characteristic of microservices, was not included in the study.

There was a study [25] wherein the standard deviation was reduced through log calculation, considering the large variance due to the nature of the resource data; and the data were learned using the LSTM model after preprocessing through the Savitzky–Golay (SG) filter. In other study [26], Bidirectional Grid (BiGrid)-LSTM, which combines Bi-LSTM and Grid-LSTM, was used for prediction. However, the existing machine learning and LSTM series are somewhat poor in long-term prediction and suffer from the disadvantage of requiring some time to generate the prediction result of the deep learning model.

The time series techniques used in existing resource prediction methods have been known to suffer from decreased performance in long-term predictions [27], and the inference process has also been found to be somewhat time-consuming [26]. Studies have been conducted on the development of the Informer model, which reduces the temporal cost of the inference process and improves the accuracy of long-term predictions [27], as well as on the use of the Informer model to predict RPC traffic in microservices for extended periods [29].

In this paper, we propose a cloud data preprocessing module that directly scrapes Kubernetes data where microservices are deployed using the Prometheus Client API, and refines the data to create a dataset that AI can learn from. After obtaining AIOps learning data through the cloud data preprocessing module, we utilized a Transformer-based Informer model, which is advantageous for long-term predictions and has shorter inference time compared to other models, to create and verify a resource prediction model.

3. Method

The goal of this study is to collect the Metric/Log data of the Kubernetes environment where micro-services are distributed and to create custom datasets that can predict the resource/disability of Pod units by branching data by service. We have developed a cloud data refining module that allows for easy extraction of data in the cloud, aggregation of the data to fit the analysis target, and creation of a dataset that can be used for AI model training.

In the existing Resource Prediction study, data is not extracted from clusters but rely on benchmark datasets such as existing Google cluster datasets. The problem of the dataset is that it is refined data differently from the metrics extracted from the general Kubernetes. To apply a model trained on benchmark data to a Kubernetes environment, the extracted data must be preprocessed to match the benchmark dataset before training can proceed. Furthermore, to apply the generated predictions to an actual service, the preprocessing steps must be repeated, which can be cumbersome. In this study, we propose a methodology for directly extracting data from a Kubernetes environment and generating cloud-based microservice analysis datasets. Using this methodology, we can extract metric data, create AIOps training datasets, and provide suitable predictions for the service.

This study involves the process of extracting and processing data from cloud clusters. After collecting data from collectors and metric or log data from each cluster, the AI model could be preprocessed, and learning could be done. The preprocessing modules were divided into three main parts: Cloud Data Scraper, Metric Integration, and Service Division. The “Cloud Data Scraper” collected all the data provided by the cluster, and because the data were metric-oriented data, there were multiple duplicate timestamps when they were collected in service units. In “Metric Integration,” the corresponding problem could be resolved, and the metric data generated in the overlapped time stamp were collected. Services that were tied up in the integrated state during the gathering process were divided using “Service Division.” The different modules are shown in **Fig. 2**. Example of generating data suitable for learning through duplicate data generation and preprocessing during the cloud metric data collection process are shown in **Fig. 3**.

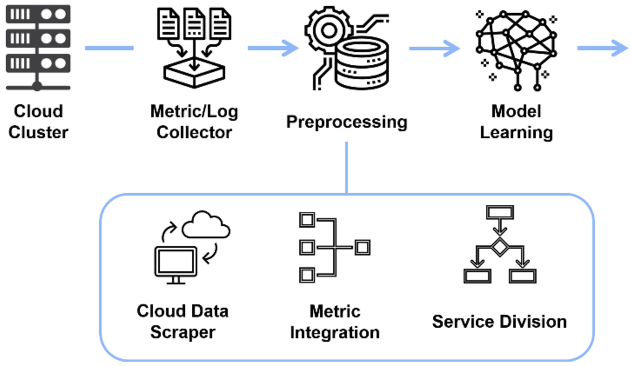
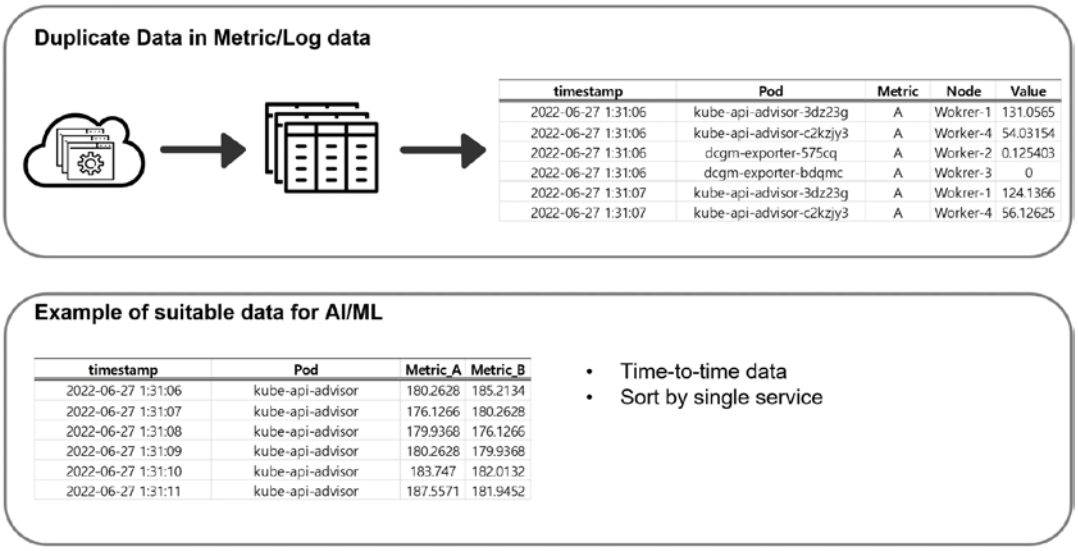


Fig. 2. Working process of cloud data refining module.


Duplicate Data in Metric/Log data
Example of suitable data for AI/ML

timestamp	Pod	Metric	Node	Value
2022-06-27 1:31:06	kube-api-advisor-3dz23g	A	Wokrer-1	131.0565
2022-06-27 1:31:06	kube-api-advisor-c2kzjy3	A	Worker-4	54.03154
2022-06-27 1:31:06	dcgm-exporter-575cq	A	Worker-2	0.125403
2022-06-27 1:31:06	dcgm-exporter-bdqmc	A	Wokrer-3	0
2022-06-27 1:31:07	kube-api-advisor-3dz23g	A	Wokrer-1	124.1366
2022-06-27 1:31:07	kube-api-advisor-c2kzjy3	A	Worker-4	56.12625

timestamp	Pod	Metric_A	Metric_B
2022-06-27 1:31:06	kube-api-advisor	180.2628	185.2134
2022-06-27 1:31:07	kube-api-advisor	176.1266	180.2628
2022-06-27 1:31:08	kube-api-advisor	179.9368	176.1266
2022-06-27 1:31:09	kube-api-advisor	180.2628	179.9368
2022-06-27 1:31:10	kube-api-advisor	183.747	182.0132
2022-06-27 1:31:11	kube-api-advisor	187.5571	181.9452

- Time-to-time data
- Sort by single service

Fig. 3. Duplicate data and suitable data for AI/ML.

The proposed cloud data refining module was configured to purify the Prometheus metric data in the Kubernetes environment. The overall structure of the module is shown in **Fig. 4**. The Prometheus Data Scraper, based on the Prometheus Client API that supports the PromQL API, collects data and the Metric Integration Module aggregates the collected data by converting it from Timestamp-centric to Service-centric and removing duplicate data. Finally, the Service Division Module separates the aggregated data by service. One advantage of the module presented in this study is that it can be easily used by anyone who owns a Kubernetes cluster with Prometheus installed because it directly extracts Metric/Log data from Kubernetes.

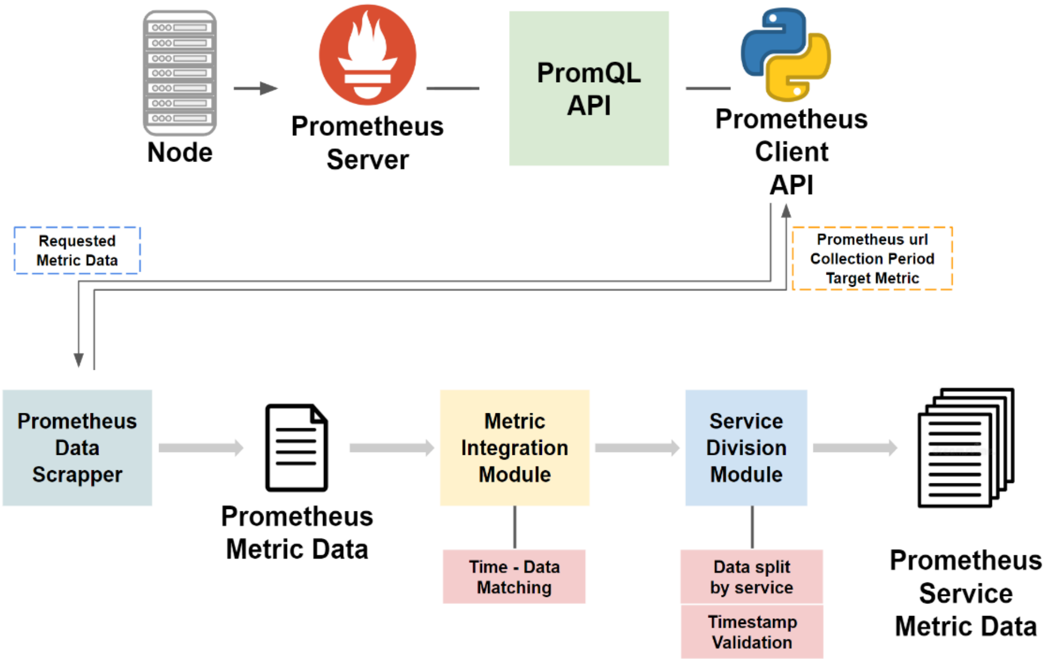


Fig. 4. Structure of cloud data refining module.

3.1 Prometheus Scrapper

Prometheus supports Prometheus Query Language (PromQL), and there is a Python "api-client-python"* that can scrap metric information through Python. This can be used in any cluster if there is an environment where Kubernetes and Prometheus are built. The Prometheus service URL, collection unit, and collection period should be entered to show the data that meet the request. Also, there is a difference in the number of metrics according to the Prometheus composition.

Prometheus data were collected using the metric comma-separated values (csv) through the package, and the data were metric-oriented because the data generated by each metric were recorded. Additionally, gathering work was needed to generate metric data for each service.

3.2 Metric Integration Module

Data were converted from metric-centered data to service-oriented data for service-specific resource prediction and failure prediction. After the timestamp, Pods, and values were extracted from each piece of metric data, they were collected in the time unit.

The timestamp of the Prometheus metric was recorded in a minute time difference for each piece of data. For example, when a metric called A was recorded in 3.05 s and one called B was recorded in 3.92 s, and the data were collected in seconds, duplicate data were generated in the same time zone. To prevent this, we adopted a method of collecting metric data that occurred below the minimum time unit. Minimum time units can be set in the units of time, namely min and s; and the collection method can be applied using the sum, average, and median.

3.3 Service Division Module

Data collected to the time unit were divided into metric data corresponding to each service. They were branched to include all the metric data recorded in the service and the missing value by service in the branching process was replaced with a very small constant value close to zero that did not interfere with the analysis. If the minimum and maximum values of the metric were missing, the metric was excluded from the analysis target.

4. Experiment

In this study, we classified the service branch units of the collected Metric data by setting them as Pods through the cloud data preprocessing module and trained using the long-term prediction capable Informer model. In addition, we compared and analyzed the extracted data with the ETTm1 experiment results used in the Informer paper to determine whether the data was suitable for the Resource Prediction task.

4.1 Data Description

In this study, we collected metrics related to the CPU and memory for resource prediction, set the collection period to one week, and confirmed that there were no obstacles due to system collision. The Kubernetes cluster comprised two master nodes and eight worker nodes, and 50 Prometheus metrics were extracted through the Prometheus Pods present at each node. After the metric data were collected, they diverged back to data corresponding to 56 Pods; the number of metrics for each Pod differed depending on whether there was a missing value or not. **Fig. 5** shows the architecture of the cluster environment.

The generated metric data were minute-to-minute data and had about 6000–10,000 lines of data per service. The kube-state-metrics (KSM) service, which was directly involved in metric collection, was selected as the subject of the experiment. The linear distribution of metric data was excluded from the single target variable. **Fig. 6** and **Fig. 7** show the metric data extracted from the KSM service according to the time flow; “container_memory_usage_bytes” follows a nonlinear distribution, whereas “container_cpu_usage_seconds_total” shows a linear increase.

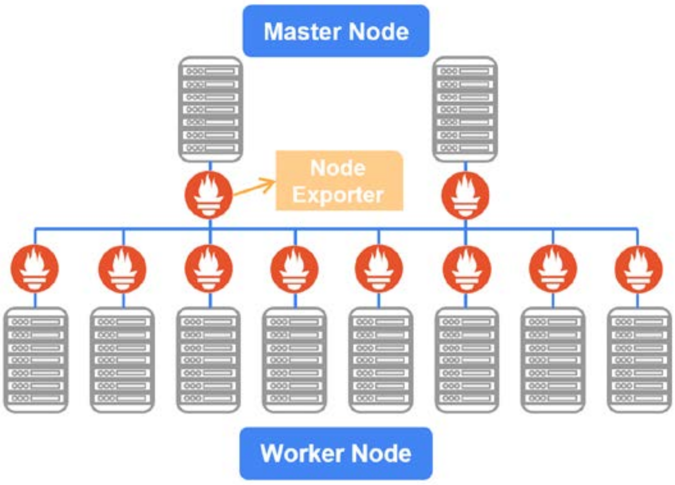


Fig. 5. Cluster environment architecture

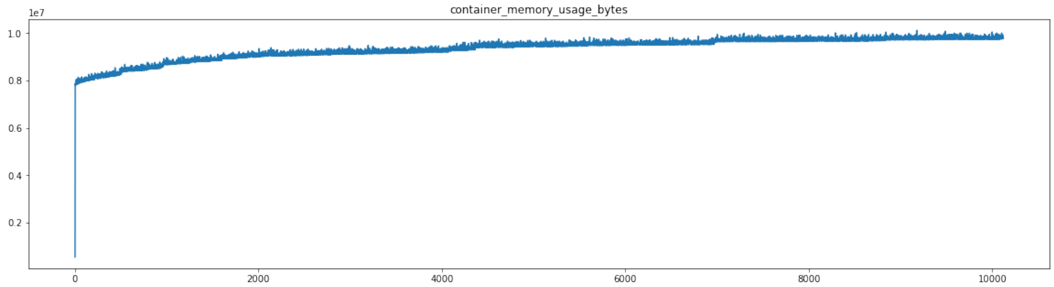


Fig. 6. KSM Pod container_memory_usage_bytes.

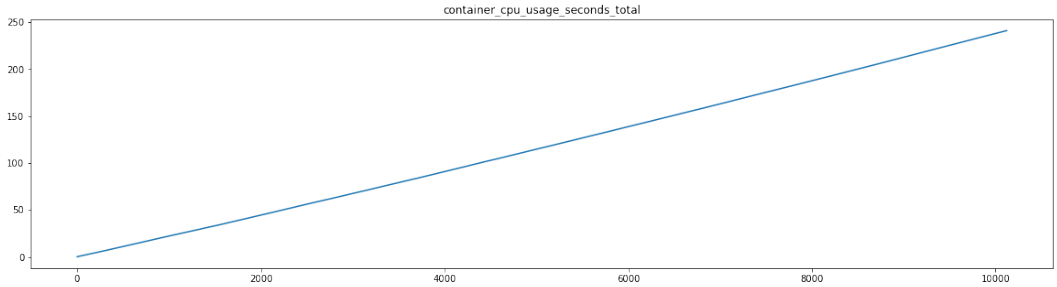


Fig. 7. KSM Pod container_cpu_usage_seconds_total.

4.2 Informer

In this study, we investigated the prediction ability of the Informer model according to two categories: multivariate prediction in multiple variables and single variable prediction in single variables. The resource prediction comparison group was set as the experimental result of the ETTm1* dataset used in the proposed paper, and the learning suitability was examined by setting the same elements, namely the input lengths of the encoder and decoder, and the prediction period. The performance was evaluated by dividing the prediction period into 24, 96, and 288 min. The optimizer applied the Adam algorithm and prevented overmatching by performing early stopping and adjusting the learning rate.

The performance evaluation was conducted by quantitative evaluation to calculate the difference between the actual metric value and the predicted metric value, and qualitative evaluation to observe the difference between the predicted value and the actual value. The MSE for measuring the accuracy of the model's result value as a quantitative index and MAE for evaluating the model's performance were used. The main evaluation factors of qualitative evaluation were how the model predicts closely with the actual value, and whether there is a statistical difference between the actual value and the predicted value.

$$\text{MSE} = \frac{1}{n}\sum_{i=1}^{n} \left(\hat{Y}_i - Y_i\right)^2 \tag{1}$$

$$\text{MAE} = \frac{1}{n}\sum_{i=1}^{n} \left|\hat{Y}_i - Y\right| \tag{2}$$

where $\hat{Y}$ means real value, Y means predicted value.

Table 1. Quantitative prediction results based on MSE and MAE.

	Univariate						**Multivariate**					
Dataset	**ETTm1**			**KSM**			**ETTm1**			**KSM**		
Pred len	24	96	288	24	96	288	24	96	288	24	96	288
MSE	0.030	0.194	0.401	0.121	**0.145**	**0.188**	0.323	0.678	1.056	0.418	**0.443**	**0.583**
MAE	0.137	0.372	0.554	0.312	**0.347**	**0.405**	0.369	0.614	0.786	0.539	**0.573**	**0.660**

4.3 Quantitative Results

Table 1 shows results of prediction experiments for single and multiple variables. In the case of a single variable, the MSE index decreased slightly compared to the existing ETTm1 Dataset when the predicted unit was 96, and the MSE decreased about twice when the predicted unit was 288. MAE indicators also showed a similar trend. In the case of multiple variables, MSE at the time of prediction 96 was significantly reduced compared to single variable, and long-term prediction showed similar trend to single variable. However, the MAE index is similar to a single variable.

In the case of short-term prediction, the index value is larger than the existing experimental results, and it is presumed that there is a difference in time series characteristics for the reason that the performance is lower than the long-term prediction. As a result of quantitative analysis, it is possible to learn, but it is necessary to verify the time series characteristics through qualitative analysis.

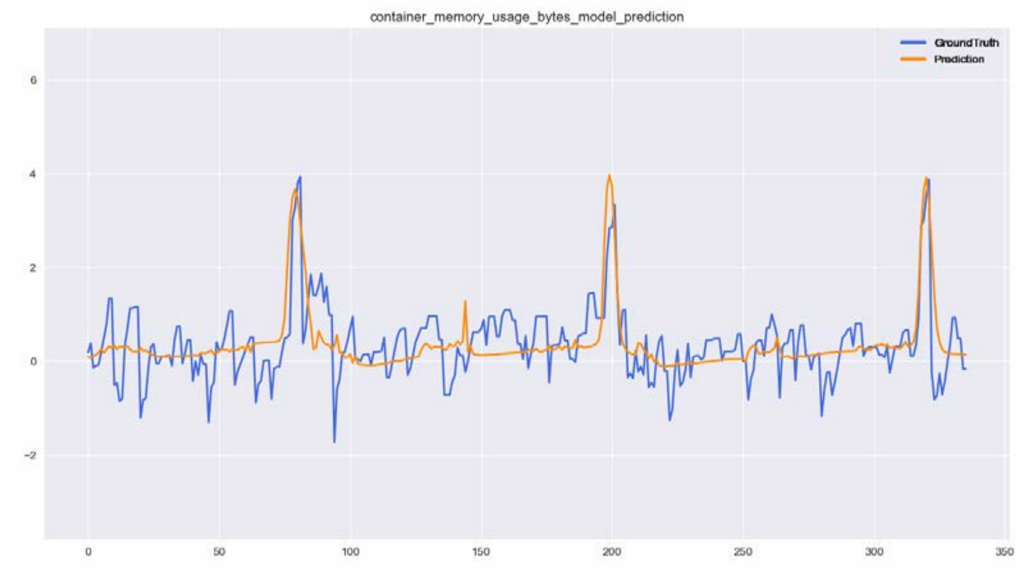


Fig. 8. KSM Univariate prediction.

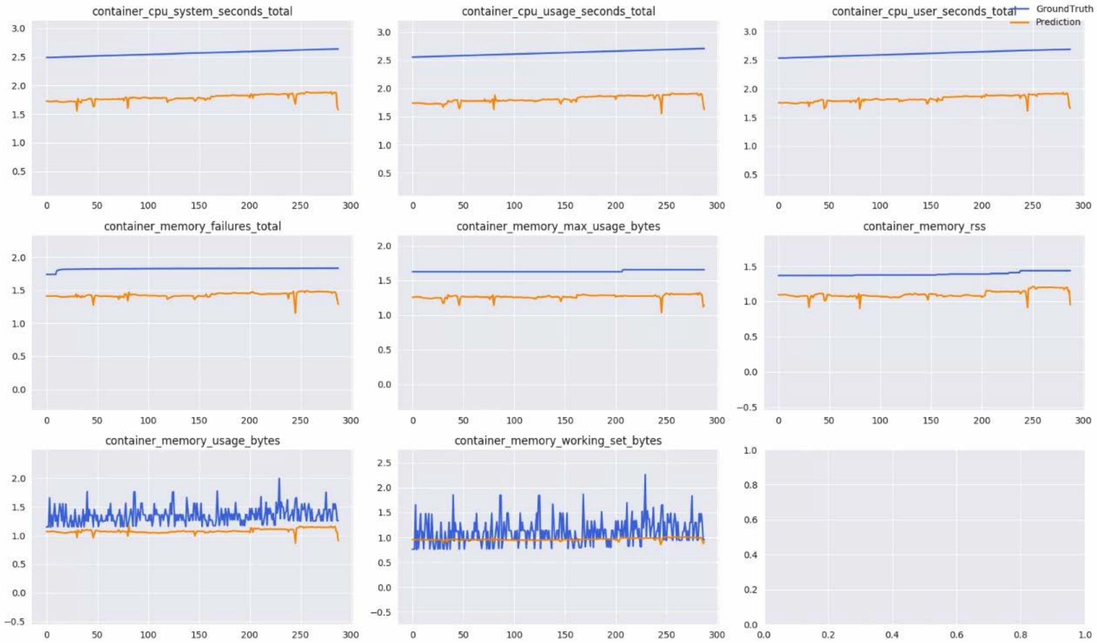


Fig. 9. KSM Multivariate prediction.

4.4 Qualitative Results

Fig. 8 compares the values of the 288 single variable prediction using the KSM with the actual values. In the section with low variation, it was confirmed that the detailed value was followed by the trend rather than predicting the detailed value, and it was confirmed that the variation was similarly predicted in the section with high variation. **Fig. 9** shows the multivariate prediction results trained by KSM dataset; and it is confirmed that it produces a prediction value with a lower variation rate than a single variable prediction result. The reason why the prediction's results for each variable have a linear distribution is that the pattern of the variables with the cumulative distribution is learned together, so it is confirmed that the linear prediction results are generated even in the variable with the volatility.

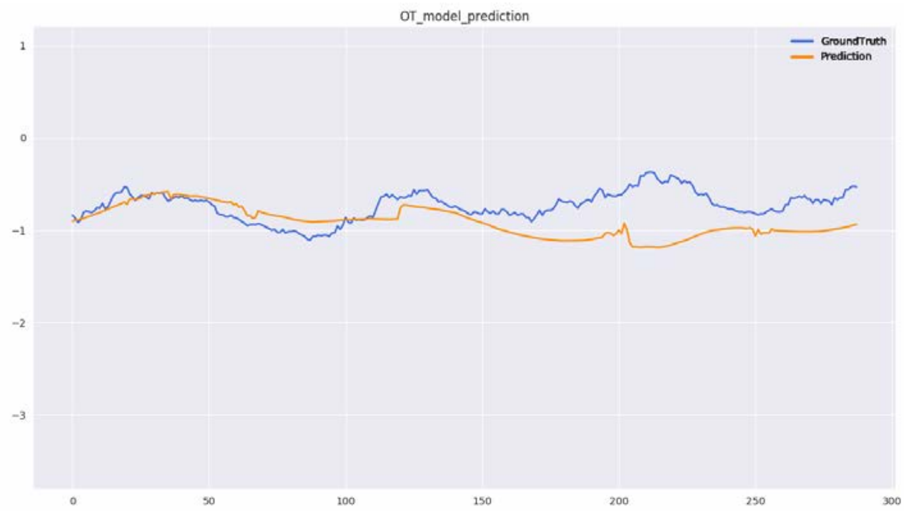


Fig. 10. ETTm1 Univariate prediction.

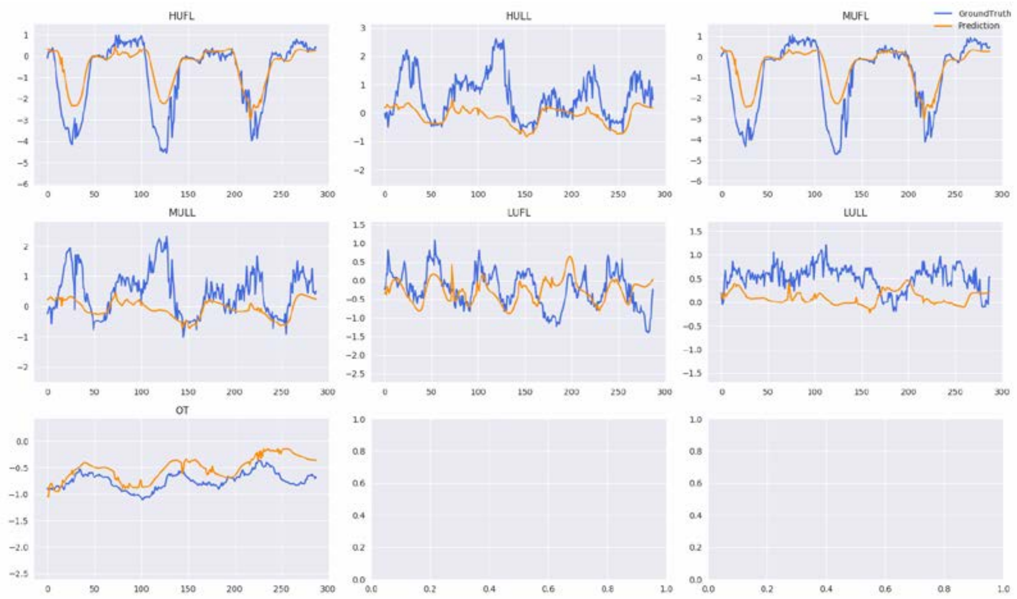


Fig. 11. ETTm1 Multivariate prediction.

Fig. 10 and **Fig. 11** represent the single variable and multivariate prediction results learned by ETTm1 dataset. ETTm1 dataset has the oil temperature as a dependent variable according to the power load and has the characteristics of analog data. Comparing the results of single variable predictions by dataset, KSM has a variety of variations and less volatility than ETTm1 datasets. This is because the target variable of ETtm1 is the oil temperature change rate of transformer, and the rate of change is lower than that of KSM dataset composed of digital signal, Metric data.

The qualitative analysis showed that the time series characteristics predicted the approximate value compared to the ETTm1 dataset, as the KSM dataset showed a smaller MSE and MAE numerical difference in the long-term prediction compared to the ETTm1 dataset. Through each experiment, it was verified that learning of resource usage prediction model of microservices learned by KSM dataset is appropriate, but it is necessary to supplement diversity of metric data to improve model performance. For this purpose, additional collection of data with various patterns is required, and a scheduler is designed to store the Prometheus data collection and preprocessing results in the database.

5. Conclusions

In this paper, we propose a module for collecting and refining Metric/Log data of cloud-based microservices for the purpose of AIOps learning. A data set was configured to input the Prometheus URL in a cloud environment to conveniently collect metric data, collect duplicate data generated in the same time zone without data loss, and then divide the data using microservices to predict the amount of resource use under each service. The Informer model was used for testing the learning suitability of the AI model using data generated through the module, and the ETTm1 dataset test results regarding the Informer model were compared and analyzed to verify the learning suitability. Although we confirmed a quantitative difference between short-term and long-term predictions, a qualitative evaluation revealed that the difference was caused by dataset characteristics, demonstrating the generation of suitable data for training microservice resource usage prediction models. Additionally, we suggest that providing long-term resource prediction values would be more helpful for users deploying microservices than providing short-term ones when attracting long-term microservices. [28]

Through this study, our contribution can be summarized as follows: 1. Our study is the first study to collect Metric data from microservice and use it for AI learning to predict resource at the service level. 2. We propose a method of preprocessing Cloud Metric Data that can be collected in cloud clusters into data that can be learned by AI. 3.The easy-to-handle module makes it easy for cloud engineers to collect and preprocess data.

As a limitation, we confirmed through experiments that the short duration of collecting cloud-based microservice data results in a monotonic distribution of metric data. Additionally, if data is collected for a long period, it is necessary to secure data through the module at regular intervals. To stably collect data, including the volatility of microservices, we have developed a module scheduler and store the results of the cloud data preprocessing module in a database. By stably expanding the preprocessed training data, we can expect to improve the performance of the resource prediction model by resolving the monotonicity issue of the distribution. As a future research direction, we plan to collect data with ensured diversity of metric data through concurrent and multiple microservice and failure scenario in cloud clusters. This will aim to improve the performance of resource prediction and failure factor prediction for each service by supplementing the limitations of Metric data distribution.

Acknowledgement

This work was supported by the Institute for Information & Communications Technology Promotion (IITP) grant that was funded by the Korea government (MSIT) (No.2021-0-00281, Development of highly integrated operator resource deployment optimization technology to maximize performance efficiency of high-load complex machine learning workloads in a hybrid cloud environment)

References

[1] Al-Debagy and P. Martinek, “A comparative review of microservices and monolithic architectures,” in *Proc. of IEEE CINTI*, Budapest, Hungary, pp. 149–154, 2018. Article (CrossRef Link).

[2] S. Taherizadeh and M. Grobelnik, “Key influencing factors of the Kubernetes auto-scaler for computing-intensive microservice-native cloud-based applications,” *Adv. Eng. Softw*., vol. 140, pp. 102734, Feb. 2020. Article (CrossRef Link).

[3] Megargel, V. Shankararaman, and D. K. Walker, “Migrating from monoliths to cloud-based microservices: A banking industry example,” in *Software Engineering in the Era of Cloud Computing*, 1st ed., pp. 85-108, Germany: Springer, 2020. Article (CrossRef Link).

[4] A. Khaleq and I. Ra, “Cloud-based disaster management as a service: A microservice approach for hurricane twitter data analysis,” in *Proc. of IEEE GHTC*, San Jose, CA, USA, pp. 1–8, 2018. Article (CrossRef Link).

[5] L. Liu, X. He, Z. Tu, and Z. Wang, “Mv4ms: A spring cloud based framework for the co-deployment of multi-version microservices,” in *Proc. of IEEE SCC*, Beijing, China, pp. 194–201, 2020. Article (CrossRef Link).

[6] M. Song, C. Zhang, and E. Haihong, “An auto scaling system for API gateway based on Kubernetes,” in *Proc. of 9th IEEE ICSESS*, Beijing, China, pp. 109–112, 2018. Article (CrossRef Link).

[7] Y. Dang, Q. Lin, and P. Huang, “AIOps: real-world challenges and research innovations,” in *Proc. of 41st IEEE/ACM ICSE-Companion*, Montreal, QC, Canada, pp. 4–5, 2019. Article (CrossRef Link).

[8] Notaro, P., Cardoso, J., Gerndt, M, “A Systematic Mapping Study in AIOps,” in *Proc. of Service-Oriented Computing – ICSOC 2020 Workshops, ICSOC 2020*, pp. 110-123, 2021. Article (CrossRef Link).

[9] Y. Lyu, H. Li, M. Sayagh, Z. M. Jiang, A. E. Hassan, “An empirical study of the impact of data splitting decisions on the performance of AIOps solutions,” *ACM Trans. Softw. Eng. Methodol.*, vol. 30, no. 4, pp. 1–38, Jul. 2021. Article (CrossRef Link).

[10] P. Townend, S. Clement, D. Burdett, R. Yang, J. Shaw, B. Slater, and J. Xu, “Invited Paper: Improving data center efficiency through holistic scheduling in Kubernetes,” in *Proc. of IEEE SOSE*, San Francisco, CA, USA, pp. 156–15610, 2019. Article (CrossRef Link).

[11] Q. Liu, E. Haihong, and M. Song, “The design of multi-metric load balancer for Kubernetes,” in *Proc. of IEEE ICICT*, Coimbatore, TN, India, pp. 1114–1117, 2020. Article (CrossRef Link).

[12] N. Sukhija and E. Bautista, “Towards a framework for monitoring and analyzing high performance computing environments using Kubernetes and Prometheus,” in *Proc. of IEEE SmartWorld/SCALCOM/UIC/ATC/CBDCom/IOP/SCI*, Leicester, UK, pp. 257–262, 2019. Article (CrossRef Link).

[13] S. Hirai, T. Tojo, S. Seto, and S. Yasukawa, “Automated provisioning of cloud-native network functions in multi-cloud environments,” in *Proc. of 6th IEEE NetSoft*, Ghent, Belgium, pp. 1–3, 2020. Article (CrossRef Link).

[14] F. A. Wiranata, W. Shalannanda, R. Mulyawan, and T. Adiono, “Automation of virtualized 5G infrastructure using mosaic 5G operator over Kubernetes supporting network slicing,” in *Proc. of 14th IEEE TSSA*, Bandung, Indonesia, pp. 1–5, 2020. Article (CrossRef Link).

[15] Perdanaputra and AI Kistijantoro, “Transparent tracing system on gRPC based microservice applications running on Kubernetes,” in *Proc. of 7th IEEE ICAICTA*, Tokoname, Japan, pp. 1–5, 2020. Article (CrossRef Link).

[16] D. Saxena and A. K. Singh, "Workload forecasting and resource management models based on machine learning for cloud computing environments," *arXiv preprint arXiv:2106.15112*, 2021. Article (CrossRef Link).

[17] F. J. Baldan, S. Ramirez-Gallego, C. Bergmeir, F. Herrera, and J. M. Benitez, “A forecasting methodology for workload forecasting in cloud systems,” *IEEE Trans. Cloud Comput.*, vol. 6, no. 4, pp. 929–941, Dec. 2018. Article (CrossRef Link).

[18] Khan, X. Yan, S. Tao, and N. Anerousis, “Workload characterization and prediction in the cloud: A multiple time series approach,” in *Proc. of IEEE NOMS*, Maui, HI, USA, pp. 1287–1294, 2012. Article (CrossRef Link).

[19] Song, Y. Yu, Y. Zhou, Z. Wang, and S. Du, “Host load prediction with long short-term memory in cloud computing,” *J. Supercomput.*, vol. 74, no. 12, pp. 6554–6568, Dec. 2018. Article (CrossRef Link).

[20] M. P. J. Kuranage, L. Nuaymi, A. Bouabdallah, T. Ferrandiz and P. Bertin, “Deep learning based resource forecasting for 5G core network scaling in Kubernetes environment,” in *Proc. of 2022 IEEE 8th International Conference on Network Softwarization (NetSoft)*, Milan, Italy, pp. 139-144, 2022. Article (CrossRef Link).

[21] T. Subramanya and R. Riggio, “Centralized and Federated Learning for Predictive VNF Autoscaling in Multi-Domain 5G Networks and Beyond,” *IEEE Transactions on Network and Service Management*, vol. 18, no. 1, pp. 63-78, March 2021. Article (CrossRef Link).

[22] Malik, S.; Tahir, M.; Sardaraz, M.; Alourani, A., “A Resource Utilization Prediction Model for Cloud Data Centers Using Evolutionary Algorithms and Machine Learning Techniques,” *Appl. Sci.*, 12, 2160, 2022. Article (CrossRef Link)

[23] D. Saxena and A. K. Singh, “VM Failure Prediction based Intelligent Resource Management Model for Cloud Environments,” in *Proc. of 2022 Second International Conference on Power, Control and Computing Technologies (ICPC2T)*, Raipur, India, pp. 1-6, 2022. Article (CrossRef Link).

[24] Erdei, R., Toka, L., “Minimizing Resource Allocation for Cloud-Native Microservices,” *J Netw Syst Manage*, vol. 31, p. 35, 2023. Article (CrossRef Link).

[25] J. Bi, S. Li, H. Yuan, Z. Zhao, and H. Liu, “Deep neural networks for predicting task time series in cloud computing systems,” in *Proc. of 16th IEEE ICNSC*, Banff, AB, Canada, pp. 86–91, 2019. Article (CrossRef Link).

[26] J. Bi, S. Li, H. Yuan, and M. C. Zhou, “Integrated deep learning method for workload and resource prediction in cloud systems,” *Neurocomputing*, vol. 424, pp. 35–48, Feb. 2021. Article (CrossRef Link).

[27] H. Zhou, S. Zhang, J. Peng, S. Zhang, J. Li, H. Xiong, and W. Zhang, “Informer: Beyond efficient transformer for long sequence time-series forecasting,” in *Proc. of AAAI*, 2021. Article (CrossRef Link).

[28] S. Luo et al., “An In-Depth Study of Microservice Call Graph and Runtime Performance,” *IEEE Transactions on Parallel and Distributed Systems*, vol. 33, no. 12, pp. 3901-3914, 1 Dec. 2022. Article (CrossRef Link).

[29] Jiyu Chen, Heqing Huang, and Hao Chen, “Informer: irregular traffic detection for containerized microservices RPC in the real world,” in *Proc. of the 4th ACM/IEEE Symposium on Edge Computing (SEC '19), Association for Computing Machinery*, New York, NY, USA, 389–394, 2019. Article (CrossRef Link).

Jonghwan Park is studying at the Institute of Electronic Components and has received a master's degree in data science from Seoul National University of Science and Technology. His research interests include conducting research on cloud systems, AIOps, and computer vision in artificial intelligence.

Jaegi Son received a PhD in Computer Science Engineering from Hankuk University of Foreign Studies. He is currently working as a Chief Researcher at Korea Electronics Technology Institute (KETI). His current research interests include real-time systems, embedded systems, clouds, and microservices.

Dongmin Kim received a PhD in Computer Science Engineering from Hankuk University of Foreign Studies in 2018. He is currently working as a Senior Researcher at Korea Electronics Technology Institute (KETI). His current research interests include real-time operating systems, embedded systems, design pattens, microservices, and cloud (SaaS) systems.