electronics

Article

Design and Implementation of Machine Learning-Based Fault Prediction System in Cloud Infrastructure

Hyunsik Yang and Younghan Kim *

School of Electronic Engineering, Soongsil University, Seoul 06978, Republic of Korea
* Correspondence: younghak@ssu.ac.kr; Tel.: +82-02-820-0841

Abstract: The method for ensuring availability in an existing cloud environment is primarily a metric-based fault detection method. However, the existing fault detection method makes it difficult to configure the environment as the cloud size increases and becomes more complex, and it is necessary to accurately understand the metric in order to use the metric accurately. Furthermore, additional changes are required whenever the monitoring environment changes. In order to solve these problems, various fault detection and prediction methods based on machine learning have recently been proposed. The machine learning-based fault detection and recovery model most commonly proposed in the cloud environment is a supervised machine learning method that learns data relating to fault situations and, based on this data, detects faults. However, there is a limit to fault learning because it is difficult to obtain all of the fault situation data necessary to learn all of the fault situations that occur in a large-scale cloud environment. In addition, it is difficult to detect a fault when a fault that differs from the learned fault pattern occurs. Furthermore, it is necessary to discuss the automatic recovery architecture leading to the fault recovery procedure based on the fault detection results. Therefore, in this paper, we designed and implemented a whole system that predicts faults by detecting fault situations using the anomaly detection method.

Keywords: cloud; availability; machine learning; fault detection; anomaly detection

Citation: Yang, H.; Kim, Y. Design and Implementation of Machine Learning-Based Fault Prediction System in Cloud Infrastructure. *Electronics* **2022**, *11*, 3765. https://doi.org/10.3390/electronics11223765

Academic Editors: Sheng Zhang, Yibo Jin, Bolei Zhang and Xiangjie Kong

Received: 26 October 2022
Accepted: 15 November 2022
Published: 16 November 2022

1. Introduction

In the cloud environment, various frameworks for detection and recovery of faults have been proposed, and most of the proposed architectures use a fault detection method based on a defined metric [1–4]. The existing fault detection architecture, which uses a metric based monitoring tool to utilize the defined metric has the following problems [5]: First, as the cloud grows in size and complexity, it is difficult to configure a fault detection environment. The cloud can take a variety of forms, including a VM-based cloud, a container cloud, and a mixed cloud. In addition, as the number of containers or VMs increases, the architecture becomes more complex and increases the target monitoring space [6–9]. In the case of the fault detection method using metrics, as in the above architecture, the administrator must individually set the threshold for each metric, and the administrator is required to have a deep understanding of all metrics. However, as the type of log data increases, the configuration becomes more and more complicated. In addition, the administrator should consider that each metric value has various types of correlation to prevent fault through an alarm at an accurate threshold value.

On the other hand, studies on cases of applying machine learning or deep learning to fault management and prediction have continuously been studied [10–19]. In particular, recently, various studies using anomaly detection methods have been conducted. In [15,16], a method for detecting faults using anomaly detection in an SDN environment has been proposed, and in [14,17–19], fault detection and prediction using anomaly detection based on cloud log data was proposed. However, because [15,16] uses SVM (Support Vector Machine), it has issues such as imbalance problems and a labeling problem. In [14,17], the PCA (Principal Component Analysis) technique is used, but it has a disadvantage as it is difficult to identify the detailed cause of the faults due to the feature of PCA. In addition, [18,19] used the Bi-LSTM technique, but this technique also has a disadvantage as labeling is difficult.

In [13], a cloud environment was constructed, and a fault detection model and a fault prediction model has been proposed by learning the fault situation for a specific resource (CPU/Memory/Network). Although [13] proposed a fault detection and prediction model for each resource using various machine learning techniques, supervised learning was used. Supervised learning-based fault detection and prediction models may also have problems with the accuracy of training data in addition to the above-mentioned labeling problems. For example, the supervised learning-based fault detection and prediction model has the disadvantage that it can detect and predict faults only for learned situations, making it difficult to detect fault patterns in fault situations that have not occurred before. Moreover, it is also impossible to create and learn all of the fault situations. In addition to this, it is also necessary to design the entire framework that controls the fault detection system and recovery system in order to start the recovery process automatically after fault detection procedures.

In this work, we designed a model that predicts cloud faults using the Self-Supervised model and developed the entire framework for fault recovery. The proposed framework is composed of two parts: The first part is the fault detection and prediction function, which consists of a monitoring data collection and processing unit and a fault notification unit. The second part is the fault recovery function, which provides a function to manage the recovery procedure by receiving fault data from the fault detection and prediction function. The proposed framework was implemented with open source cloud platforms, such as OpenStack and Kubernetes, to verify the function [20,21].

The contents of this paper are as follows. In Section 2, we analyze previous works relating to machine learning-based fault management and recovery system frameworks in the cloud environment, and in Section 3, we describe the proposed architecture and machine learning theories we used in this paper. In Section 4, we verify the performance through the actual implementation environment, based on the proposed architecture, and we conclude in Section 5.

2. State of the Art: Machine Learning Based Fault Detection and Fault Prediction

In a cloud environment, cloud management functions such as MANO provide availability for the entire infrastructure and services [22]. In order to guarantee availability in a cloud environment, information on the entire infrastructure and services, and a recovery function based on the information, must be provided. Put simply, in the cloud environment, a monitoring tool is used to detect faults, and a threshold value for each metric is set, and the case of exceeding the threshold value is determined as a fault [10–13].

However, it is difficult to configure such static policy-based management functions as the cloud size and its complexity increases. In addition, the understanding of individual values for all metrics and accurate set values are required, and values must be changed according to circumstances. To solve this problem, machine learning-based fault detection models have recently been proposed.

In [13], the researchers designed and validated an architecture for machine learning-based fault detection and prediction in a cloud environment: a cloud environment, consisting of a center cloud and an edge cloud, was implemented using Kubernetes and OpenStack, and then an architecture for a fault detection model and prediction was designed and applied to the cloud and verified. In [13], various fault detection models and deep learning models were used to verify the accuracy of the proposed architecture. To implement fault situations, CPU, memory, and HDD failures were artificially generated to generate data on failure conditions, and the accuracy of the proposed model was measured based on the generated data.

However, the supervised learning-based method has the following limitations:

Firstly, data generated in an actual fault situation is not used for model learning. In the previous work, data was generated by artificially generating a fault in order to learn the fault situation. As this data was used for fault detection and predictive learning, it will be easy to detect faults because the pattern is almost the same as the learning data. In other words, the data generated for fault learning and the data used to detect the fault situation in the previous study are the same. However, in an environment where the cloud is actually operated, various types of fault patterns can exist; therefore, it is difficult to determine actual faults through learning using artificially generated data.

Secondly, the number of monitoring data resources used in the cloud environment consist of thousands to tens of thousands of types. This is constantly changing each time the cloud environment is operated, and the pattern of change is different for each resource. As in previous studies, supervised machine learning methods require learning about a specific resource, and this requires knowing which resource changes to apply in the event of a fault. However, as the size of the cloud increases, it becomes difficult to ascertain the changing nature of resources or the correlation between resources. In addition, in order to analyze the fault of various resources, a detailed analysis of the data is required because the labeling of the fault status for each resource is required.

Thirdly, it is difficult to accurately analyze the cause of a fault. In existing studies, because only the data classified as a fault during learning is detected as a fault, analysis is difficult when there are other causes, and it is difficult to determine various causes of a fault when it occurs due to complex causes.

Fourthly, an interworking architecture is required alongside the cloud management architecture. In order to prevent actual faults by using the results derived through fault detection and fault prediction models, it is necessary to link with the recovery system. However, previous works focus on fault detection techniques; therefore, to provide an environment that guarantees actual system availability, a design and interworking pipeline for an integrated architecture is required.

Therefore, in this paper, we designed a framework that can detect and predict faults using unsupervised learning. In addition, a function to automatically perform fault recovery based on the detection result was designed. The proposed architecture was implemented by applying it to the actual cloud environment, and the proposed model and functions were verified in the implementation environment.

3. Design and Implementation of Machine Learning-Based Fault Prediction System in Cloud Infrastructure

3.1. Proposed Architecture

Cloud infrastructure has a complex architecture. With respect to the complex architecture of the cloud infrastructure, the types of log data are also diverse, and the number of log data also changes as the number of virtual machines increases or decreases. Moreover, the average utilization rate of each resource varies according to the type of service and resources. In the cloud environment, cloud availability has been traditionally guaranteed through a monitoring system, but accurate fault detection is difficult due to the above-mentioned structural characteristics. To overcome this, fault detection techniques using supervised learning have been proposed, but the problem of labeling all data, securing data for fault learning, and detecting the causes of complex faults still remain. To overcome this, in this paper, we proposed a fault detection method using a self-supervised learning method that does not require labeling.

The proposed architecture is shown in Figure 1 And, as shown, the proposed architecture consists of two components: The first component is cloud infrastructure management, and the second component is a machine learning-based fault prediction system. The management component consists of an orchestrator for the overall infrastructure management and a fault management and recovery function. The orchestrator includes the ability to manage the VM-based cloud infrastructure and container-based cloud infrastructure.

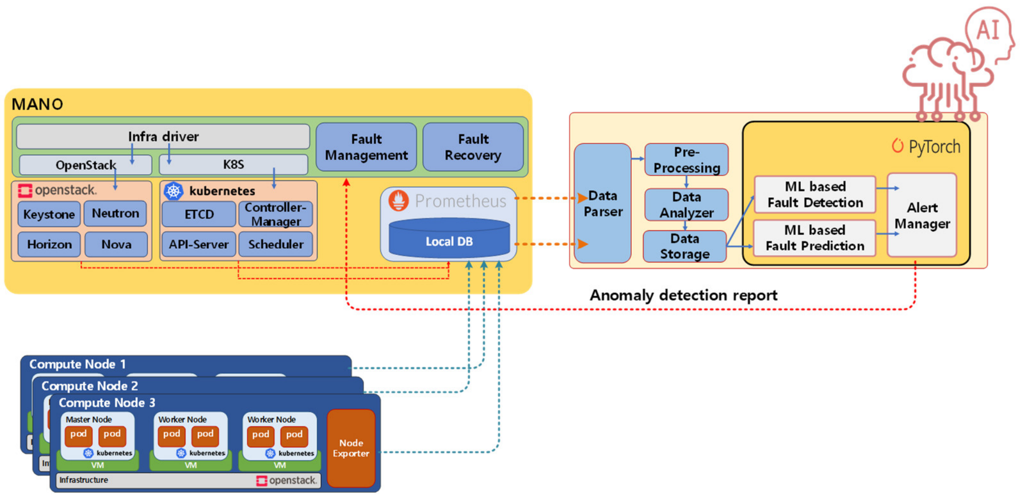


Figure 1. Proposed Architecture for ML based Fault Prediction Model.

The fault management and recovery function provides a function to check and recover from a fault, according to the fault management policy. The recovery function includes functions such as VM restart, VM respawn, and switch over. The machine learning-based fault prediction system consists of a receiving unit that receives monitoring data, a data processing unit that processes it, and a fault prediction system that checks the fault prediction value by inputting the processed data into a learned model. The data receiving unit periodically receives the current state data from the monitoring tool of the cloud infrastructure, and the received data is processed into a form suitable for the machine learning model through the data processing unit.

Subsequently, the processed data determines whether a fault has occurred through a machine learning model, and the fault information is delivered to the cloud management function.

3.2. Self-Supervised Fault Prediction

Supervised learning could not show a better performance than the existing work which used supervised method because it requires an abnormal sample for training. In the case of semi-supervised learning, anomalous samples are not required, but labeling is required. However, labeling is not suitable for data-heavy environments such as cloud environments. Therefore, self-supervised learning, a model suitable for cloud environments and that overcomes the above shortcomings, was selected as the model for this work.

The proposed architecture uses an Anomaly Detection method based on Long Short-Term Memory models (LSTM) like a Figure 2 [23–26].

For anomaly detection, a data set of a normal environment is required, and in this proposal, data is collected and learned in a cloud environment that operates normally for normal state learning. After learning, we saved the model that is trained based on the normal data, the average of the reconstruction error data of the normal data, the covariance, and the scaler. Next, the data set required for fault detection was put into the model, from which normal data was trained and the reconstruction error was calculated. The calculated reconstruction error was entered into the Gaussian distribution, created from the previously learned data, and the likelihood was calculated, and this value was defined as an anomaly score.

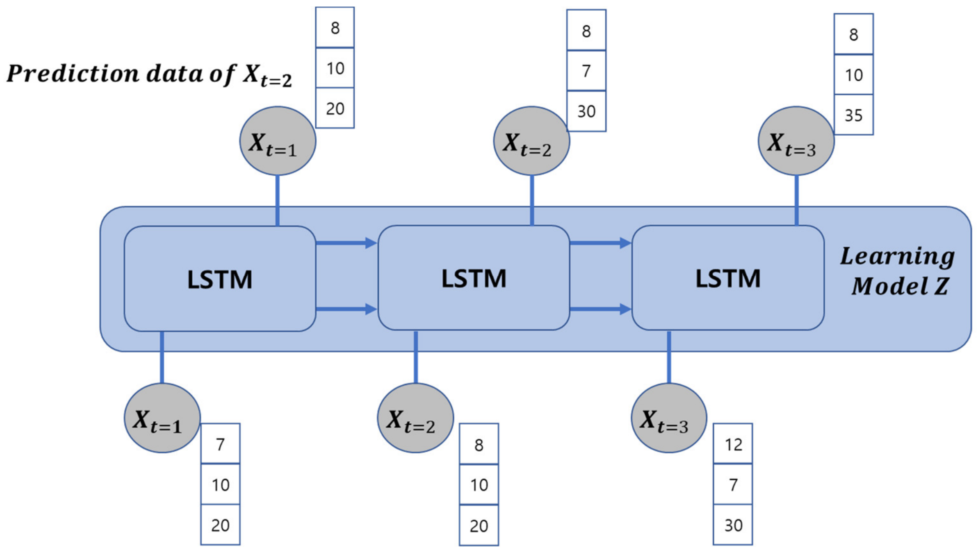


Figure 2. Self-Supervised LSTM model for Fault Prediction.

3.3. Data Pre-Processing

In order to use data for a machine learning model, data collection and processing procedures are required. In the present work, the Prometheus monitoring tool was used to check the current state of the cloud environment. Data were collected from each node, VM, and other services, once per second, and the total number of features is approximately 4600. In this study, the following data pre-processing procedure was performed to learn and predict a failure detection model. First, data was collected using PROMQL [27]. Then, a format change was performed to adjust the format of the collected data. In Prometheus, it is defined to use a Count value or a Gauge value according to the data type. Some count values were changed to gauge values to ensure the accurate analysis of the collected data. Finally, the data were changed to the "csv file" format and subsequently saved. Interpolation and missing value processing were also performed on the missing data; the Standard Scaler was used for data scaling before learning the data.

3.3.1. LSTM Based Anomaly Detection

In this study, we designed an architecture for anomaly detection with self-supervised learning. The data do not require labeling after the pre-processing process, but the data set we created and configured to determine whether fault detection is performed accurately. The data set used in this study is shown in Table 1.

Table 1. Dataset for Fault prediction.

	Data Type	**The Number of Features**	**Description**
	Normal data for training	1650	Containers which runs on the VM
Recurrent Fault	Fault Dataset (CPU)	1650	Increase CPU usage
	Fault Dataset (Memory)	1650	Increase memory usage
	Fault Dataset (Network)	1650	Increase network I/O usage
Accumulative Fault	Fault Dataset (CPU)	1650	Increase CPU usage
	Fault Dataset (Memory)	1650	Increase memory usage
	Fault Dataset (Network)	1650	Increase network I/O usage

The first data set is for learning the normal state. A cloud environment was constructed, and the data were extracted from the actual operating cloud.

In order to create a fault situation, in this work, the amount of change in the Anomaly Score, according to the change in resource usage, was measured by increasing three resources. First, the CPU usage was arbitrarily increased by using the stress-tool to assume the CPU fault environment. For the memory and network I/O, the resource usage was increased using the same tool, and data sets with increased resource usage were stored as individual data sets [28,29]. The fault data was composed of recurrent data and accumulative data, and the data were separately generated for each. In the case of the recurrent dataset, a fault was generated, periodically, for a specific period of time, and in the case of the accumulative dataset, the data set was generated by continuously increasing the stress by 20–30%. To produce a learning model, we entered normal data into the LSTM model and trained it. The model produced at this time is used for subsequent data discrimination, and the distribution of the trained model and the equation of reconstruction error from the model is as follows [24].

$$\text{Reconstruction Error } e^{(i)}, e^{(i)} = \left| X^{(i)} - X'^{(i)} \right| \tag{1}$$

$X^{(i)}$ is the value of the i-th timestamp of a specific feature, and $X'^{(i)}$ is the value of the timestamp after predicting by putting data into the trained model.

The model used in this work predicts the value of the next timestamp through the trained model and determines whether or not a fault has occurred in the current state, based on the difference with the actual value. That is, after the model is trained, it is possible to predict normal data, but it is difficult to predict abnormal data. At this time, the difference between the predicted value and the actual value is called a Reconstruction Error, and according to the difference, it is possible to determine whether the current state is a normal state or not. After the model is trained, the average and covariance matrix of the reconstruction errors obtained for each feature are stored, and the Anomaly Score can be calculated as follows [24]. The Reconstruction Error for each feature can represent the difference between the predicted value and the actual value as an absolute value. When the predicted value is $X'^{(i)}$ and the actual value is $X^{(i)}$, the Reconstruction Error can be calculated as $e^{(i)} = \left| x^{(i)} - x'^{(i)} \right|$. The average of the reconstruction error for each feature is represented by μ and calculated, such as in Equation (2), and the covariance matrix is represented by $\sum$ and expressed through Equation (3). Based on this, the anomaly score can be calculated according to Equation (4).

$$\mu = \frac{1}{m} \sum_{i=1}^{m} e^{(i)} \ (\text{m} = \text{The number of features}) \tag{2}$$

$$\sum = \frac{1}{m} \sum_{i=1}^{m} \left(e^{(i)} - \mu\right) \left(e^{(i)} - \mu\right)^{T} \tag{3}$$

$$\textbf{Anomaly Score} = \left(\boldsymbol{e^{(i)}} - \boldsymbol{\mu}\right)^{\boldsymbol{T}} \sum\nolimits^{\boldsymbol{-1}} \left(\boldsymbol{e^{(i)}} - \boldsymbol{\mu}\right) \tag{4}$$

when normal data comes into the trained model, the anomaly score is low because the probability of accurately predicting the next data is high. On the other hand, when data containing a fault comes in, because it is not training data, the prediction probability decreases and the difference in the anomaly score increases. That is, if the anomaly score for new data is low, it can be determined as normal, and if it is high, it can be determined as abnormal.

3.3.2. Fault Recovery System Based on Machine Learning

Figure 3 is the architecture for fault recovery using machine learning-based fault detection results. According to the fault detection results, appropriate fault recovery was performed, and for this purpose, a workflow manager was designed that can perform fault recovery procedures using various infrastructures. The workflow manager provides a function to perform a defined workflow according to the failure detection result and is designed to use the management function of the infrastructure.

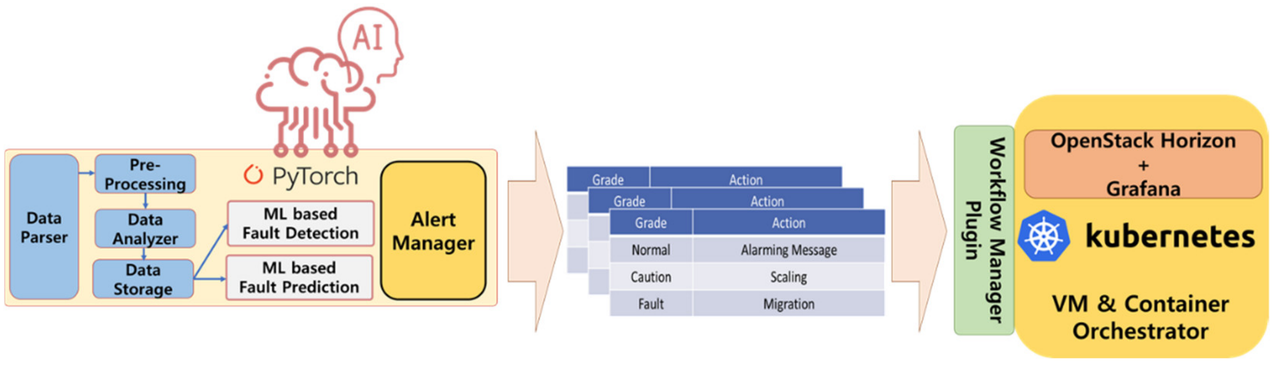


Figure 3. Machine Learning based Fault Prediction Architecture.

In this work, we designed the workflow manager to start the fault recovery procedure using the change amount and value of the anomaly Score. The workflow can be altered according to user definition, and the procedure is as follows.

Following anomaly detection, we used the feature importance model to find of the cause of the fault. By separately extracting only the abnormally detected part and applying the feature importance algorithm, the cause of the fault was found efficiently. As shown in Figure 4, the total time consists of data collection time, data processing time, prediction processing time, feature importance processing time, and recovery time. The data collection time is an interval at which is the data are collected by the actual monitoring tool and may vary depending on the setting of the monitoring tool. The data processing time is the time required to collect and process data. The prediction processing time is the time required for the preprocessed data to calculate the anomaly score in the model. The feature importance time is the time required to check which features have had a lot of influence in the dataset when an anomaly score increases.

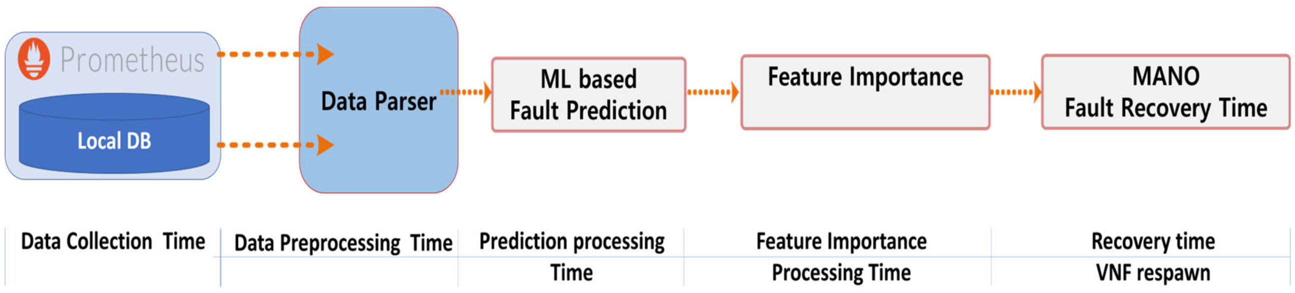


Figure 4. Workflow of Fault Recovery.

4. System Validation

4.1. Implementation

For evaluation, a test bed was implemented, as shown in Table 2. A total of four servers were used, and OpenStack and Kubernetes clusters were used as the cloud infrastructures. For the machine learning environment, one machine learning server was individually implemented, and the model was designed using Pytorch. Prometheus was used for cloud monitoring and data collection [30,31].

Table 2. Implementation specifications for container.

Entity	Condition	Version
Physical Server (4)	Controller Node (1)/Compute Node(2) Intel(R) Xeon 2.4 GHz × 80 vCPU RAM: 64 GB Disk space: 2 TB GPU Node (1) Intel(R) Xeon 3.4 GHz × 12 vCPU NVIDIA Tesla V100 32 G (4EA)	
Cloud OS	OpenStack stable	Stein
Container OS	Kubernetes	1.17.1

The collected data were stored in a control node installed with Prometheus, and a GPU server was used for data preprocessing and the learning model.

4.2. Test Result

Each anomaly value was measured using the data set mentioned in Table 1, and the results are as follows. In all of the graphs included in this chapter, the red line illustrates the artificially generated resource usage, and the pattern or amount of change can be altered depending on the data set. The blue line depicts the anomaly score.

In the case of recurrent CPU failure cases, as shown in Figure 5, it was observed that the anomaly score increased at the same time as the fault occurred. However, it was similarly observed that the anomaly score did not increase for temporary CPU changes. In other words, it was confirmed that the anomaly score did not increase for regular CPU change patterns through learning.

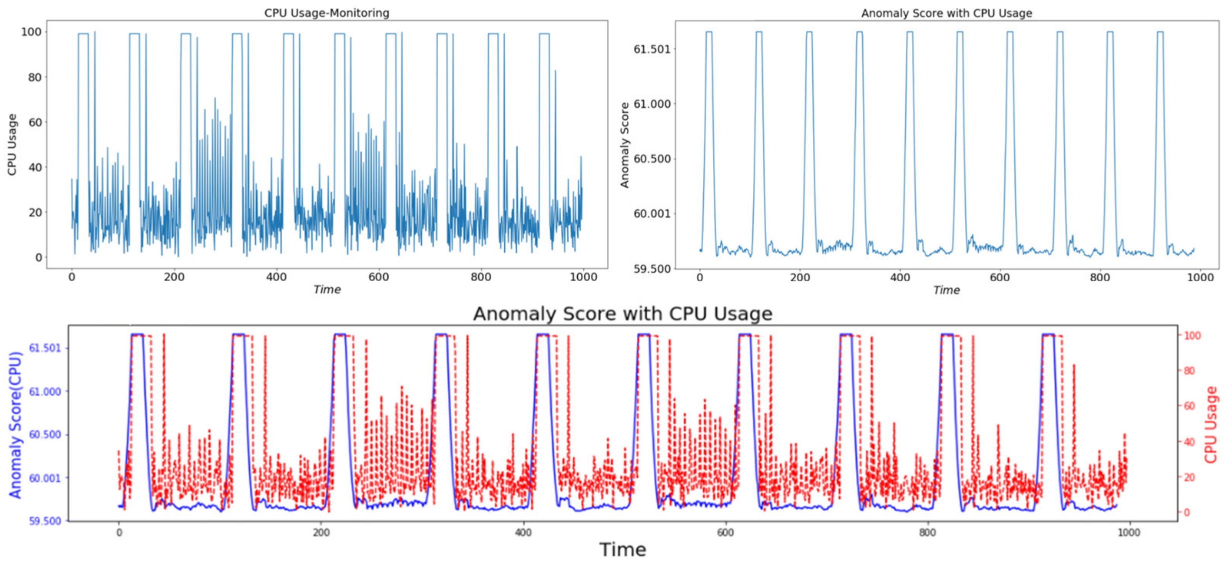


Figure 5. Anomaly Score (CPU Fault).

Next is a graph of memory fault occurrence. As shown in Figure 6, the memory usage was repeatedly increased to the maximum. In the case of a memory fault, it was confirmed that the anomaly score increased at the same time as the fault occurred.

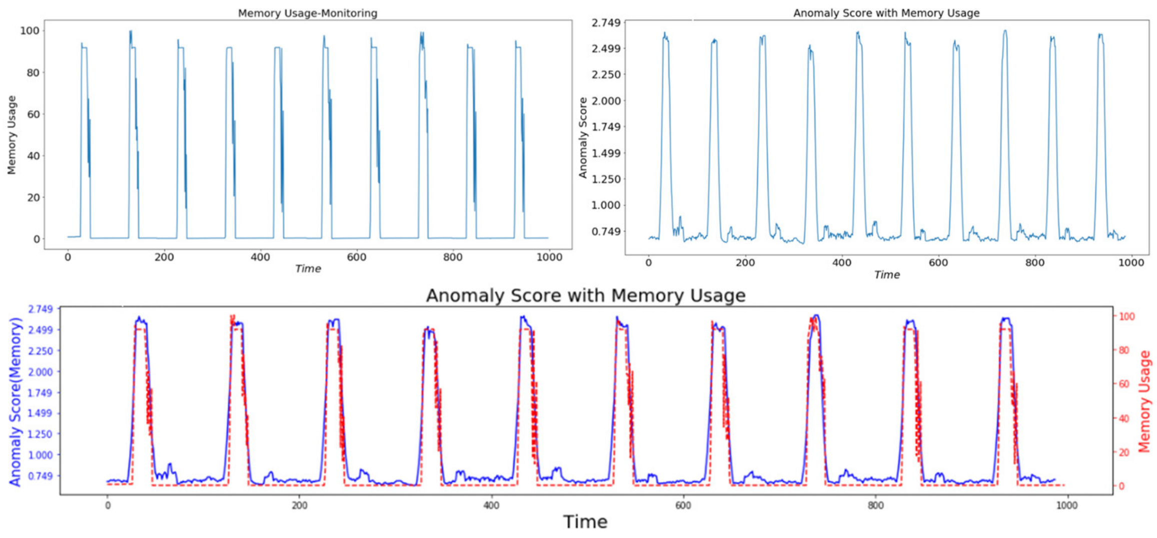


Figure 6. Anomaly Score (Memory Fault).

Subsequently, an experiment was conducted on the network fault. Network faults were created by increasing I/O, flooding UDP packets, and creating socket processes. Like a Figure 7, in the case of network failure, it was confirmed that the anomaly score increased when the fault occurred.

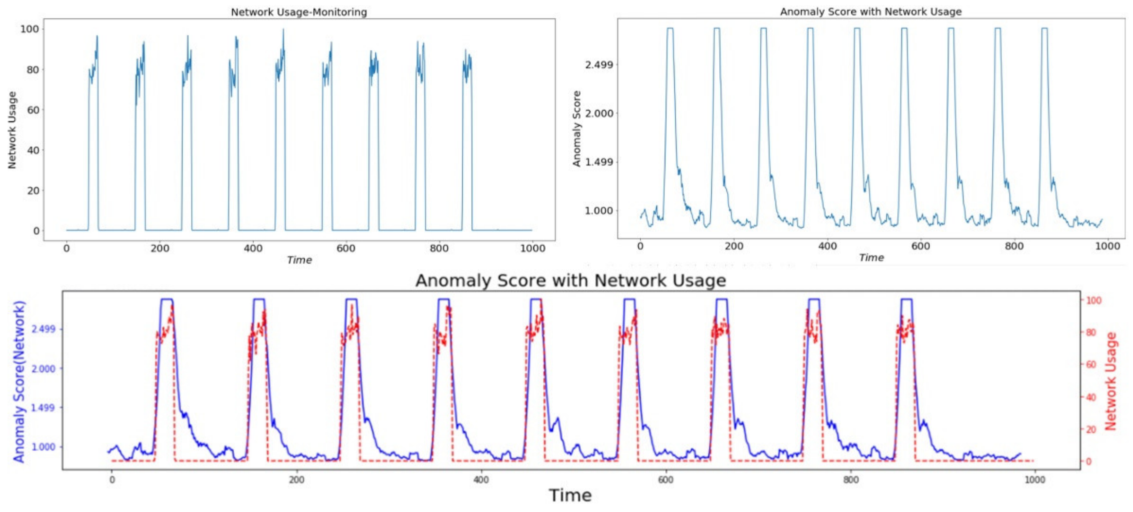


Figure 7. Anomaly Score (Network Fault).

Next is the experimental results for cases in which the fault situation gradually occurs. We used same method that was used previously, and the results are as follows.

As shown in Figure 8, with the gradual increase in CPU usage, the anomaly score gradually increased in line with this increase. It was confirmed that the anomaly score also gradually increased between the timestamps 80 and 100, which is the time that the change occurred.

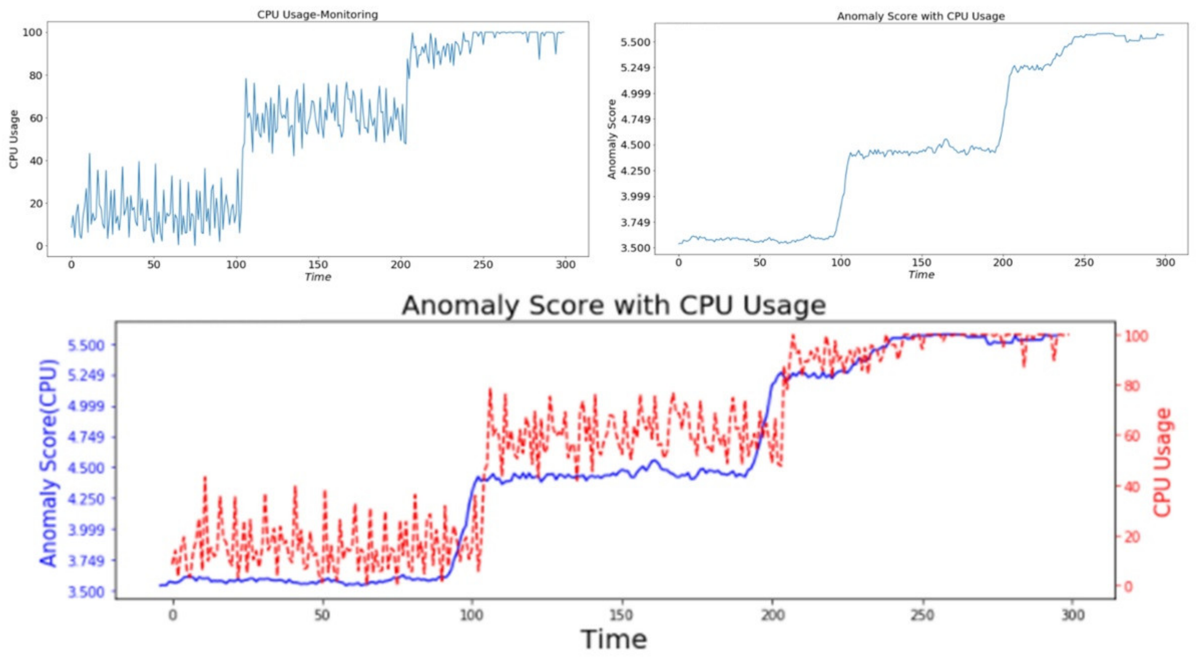


Figure 8. Anomaly Score (CPU Fault-Accumulative).

As shown in Figures 9 and 10, it was confirmed that the same results as in the case of a CPU increase were produced, even when the memory and network faults were gradually made.

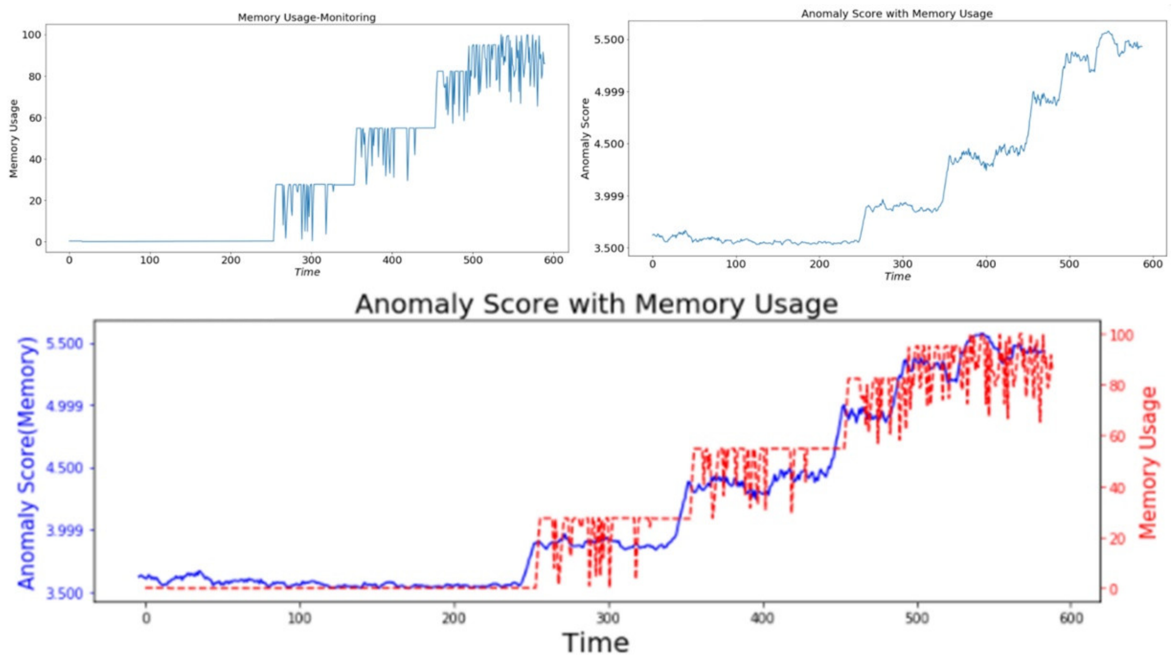


Figure 9. Anomaly Score (Memory Fault-Accumulative).

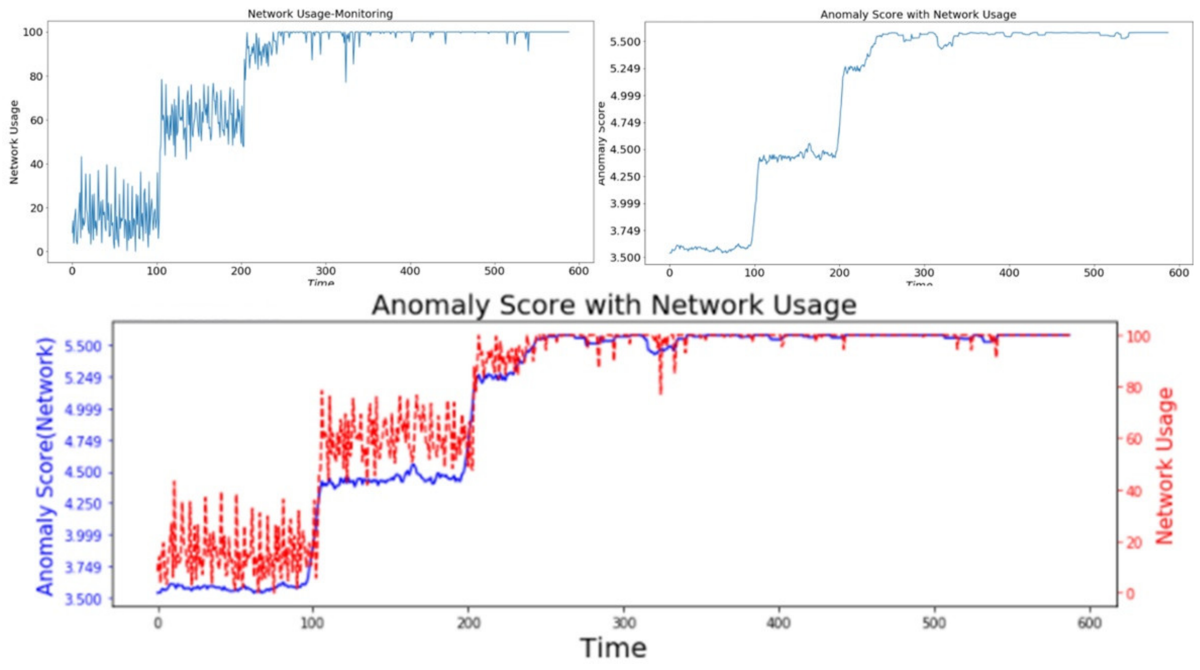


Figure 10. Anomaly Score (Network Fault-Accumulative).

In order to check the accuracy of the model, a virtual noise was created, and the detection results were plotted. As shown in Figure 11, virtual noise was synthesized in only some sections of one data extracted from the cloud environment. It was inferred that the actual anomaly score value increased for the section to which noise was applied.

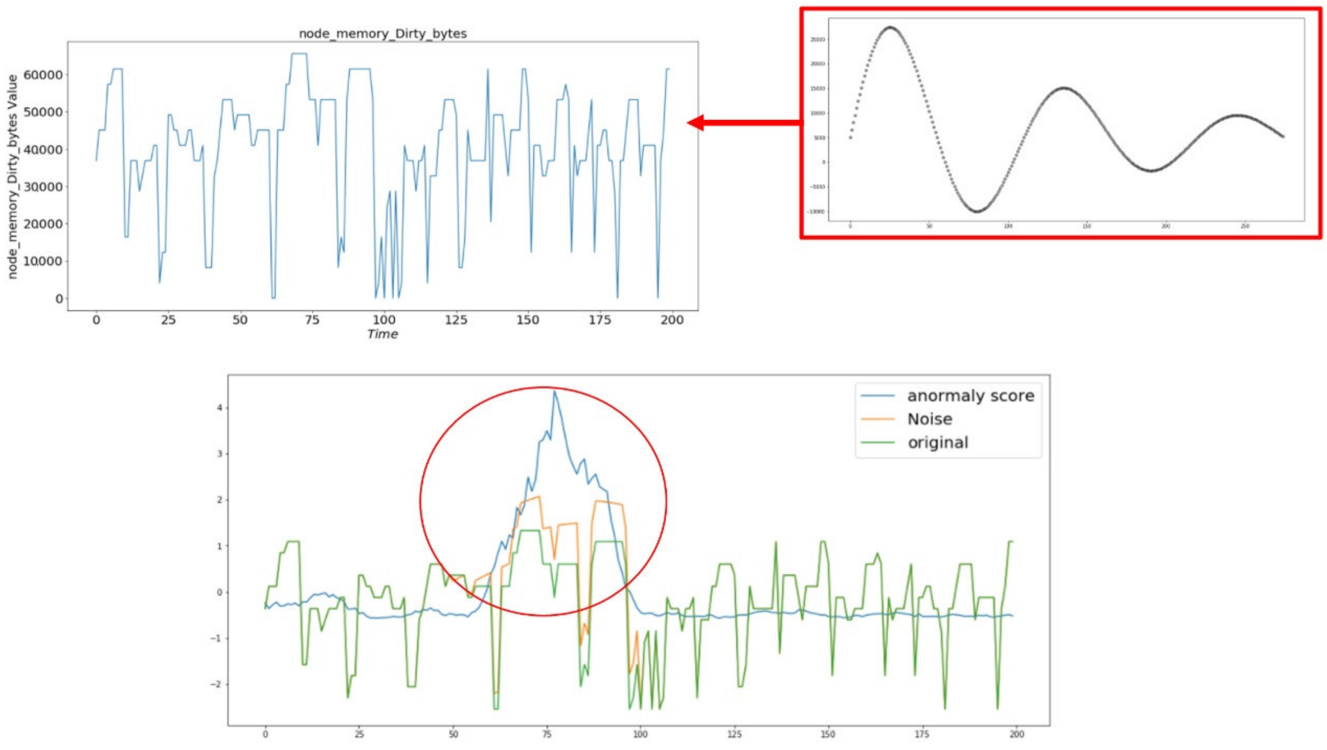


Figure 11. Anomaly Score (Virtual noise Fault).

As shown in Table 3, over 95% of the precision, recall, and F1 score were verified. However, in the case of an Accumulative Fault, it was confirmed that the precision decreased in the section in which the resource usage increased and affected the overall performance. In addition, the overall precision was higher than the recall. However, in the F1 score, considering these two, it was confirmed that overall performance was measured similarly. Next, the classification performance was measured using the ROC (Receiver Operating Characteristic) curve.

Table 3. Precision, Recall, F1 Score of LSTM Model.

	Recurrent Fault	Accumulative Fault
Precision	96.1	95.6
Recall	95	95.6
F1 Score	95.5	95.6

The precision, recall, and F1 scores for the results are as follows [32,33].

Figure 12 shows the ROC values for the test sets of Recurrent Fault and Accumulative Fault. As shown in Figure 12, it was confirmed that the overall data set was accurately classified. With Table 3 and Figure 12, it was confirmed that the overall performance of the proposed architecture was high.

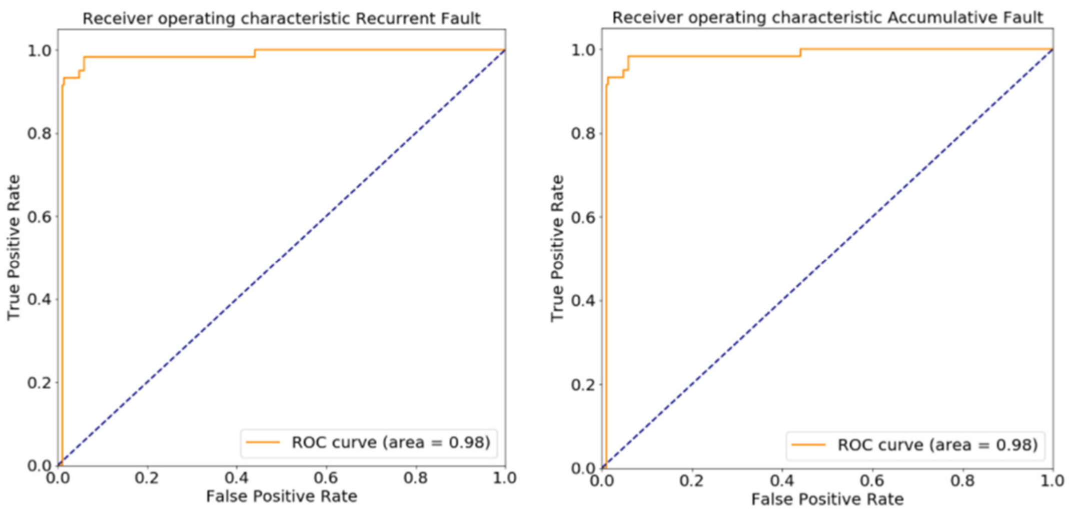


Figure 12. ROC Curve of LSTM Model.

4.3. Testbed for ML Based Cloud Infrastructure Management

Based on the model described in the previous chapter, the entire architecture that can perform the processes, from data collection to recovery, was implemented.

Figure 13 shows the architecture for measuring an anomaly score of the cloud using the learned model and calling the recovery function based on the result value. The fault prediction procedure in the proposed architecture is as follows. The data collected by the monitoring tool is forwarded to the prediction model. Once these data are received, the anomaly score is tested to confirm whether the value is higher than the specified threshold. If the value is high, check the rate of change in the anomaly score. If the rate of change is not higher than the reference value, a warning message is sent to the cloud system. On the other hand, if the anomaly score change rate is more than the value administrator defined, the fault label is added to the area where the anomaly score is high and transferred to the feature importance analysis. The feature importance analysis forward the feature list that has changed the most in the current state to the operator, and the operator checks the current status of the cloud system with the feature lists. If there is an anomaly pattern in the resource value and system, it is immediately transmitted to the recovery function of the cloud system. If not, additional learning is performed, including previous data, to include the abnormal operation in the normal data set.

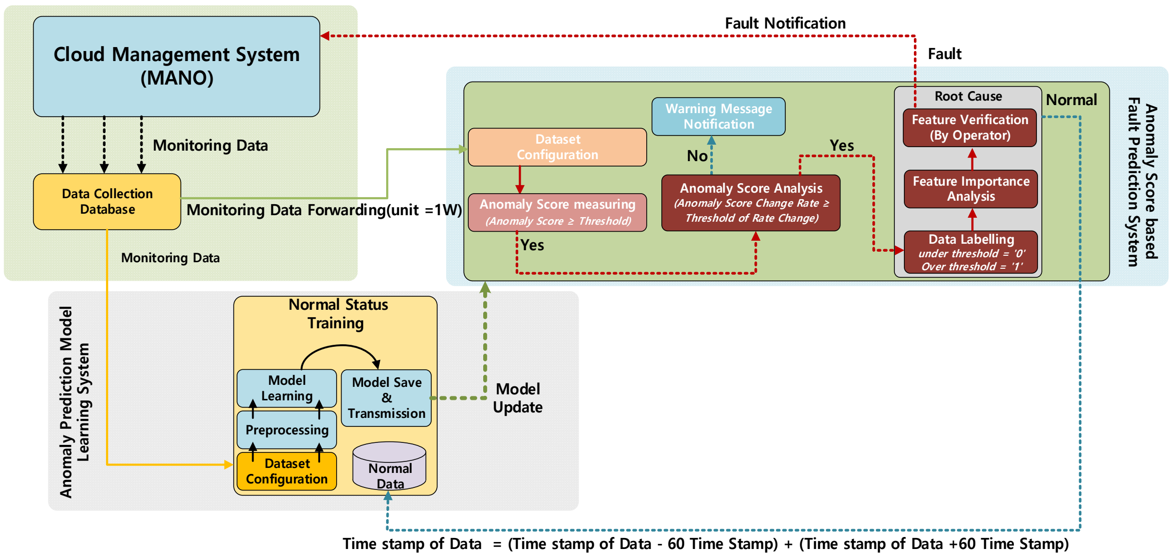

Cloud Management System (MANO)
Monitoring Data
Data Collection Database
Monitoring Data Forwarding(unit =1W)
Monitoring Data
Fault Notification
Fault
Normal
Root Cause
Feature Verification (By Operator)
Feature Importance Analysis
Data Labelling under threshold = '0' Over threshold = '1'
Warning Message Notification
Dataset Configuration
Anomaly Score measuring (Anomaly Score ≥ Threshold)
Anomaly Score Analysis (Anomaly Score Change Rate ≥ Threshold of Rate Change)
No
Yes
Yes
Anomaly Score based Fault Prediction System
Anomaly Prediction Model Learning System
Normal Status Training
Model Learning
Model Save & Transmission
Preprocessing
Dataset Configuration
Normal Data
Model Update
Time stamp of Data = (Time stamp of Data - 60 Time Stamp) + (Time stamp of Data +60 Time Stamp)

Figure 13. ML based Cloud management System.

Figure 14 is an experiment that increases the anomaly score by adding virtual noise, and tests whether the anomaly score increases. Following this test, for the purpose of cause analysis, a label was added to the area (blue) where the anomaly score increased and was transferred to the feature importance module.

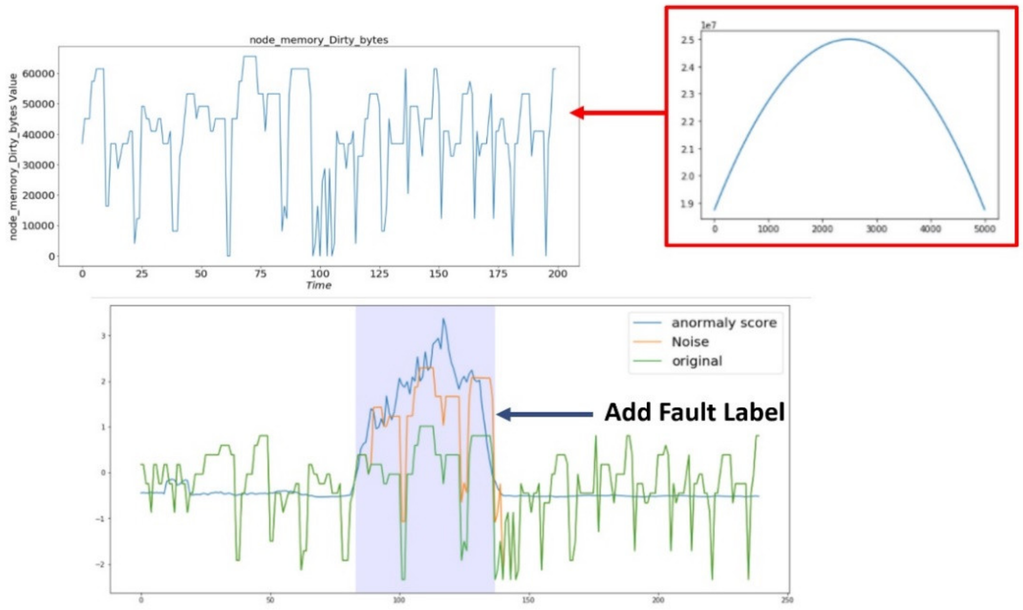


Figure 14. Feature Importance Test.

Figure 15 is the result of confirming which features have substantially changed through the feature importance module. It was confirmed using the LGBM(Light Gradient Boosting Machine)-based feature importance method, and as shown in Figure 15, it was confirmed that the feature omitting virtual noise was ranked at the top [34]. After confirming the actual feature change pattern through resource verification, if there is a problem, it is forwarded to the cloud system. If not, it is retrained by adding the data to the existing prediction model.

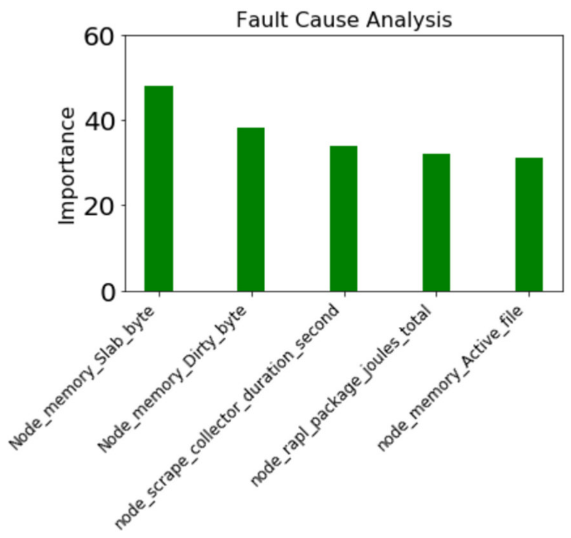


Figure 15. Feature Importance Result.

Figure 16 is the architecture for fault information transmission. The information includes the server addresses, feature importance results, and fault levels. After receiving the information, the server's workflow manager begins the recovery procedure, according to the defined policy.

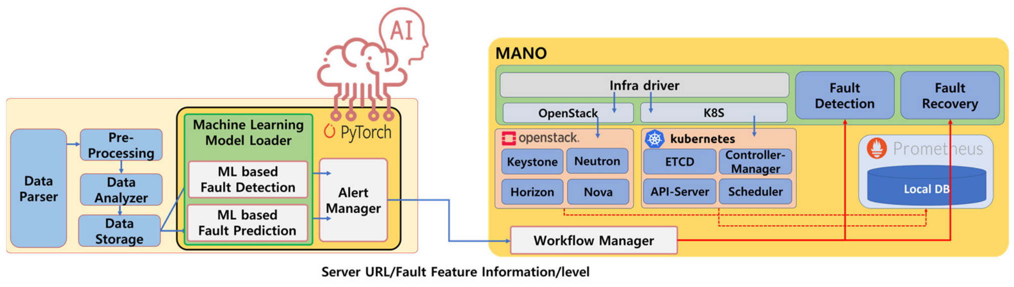
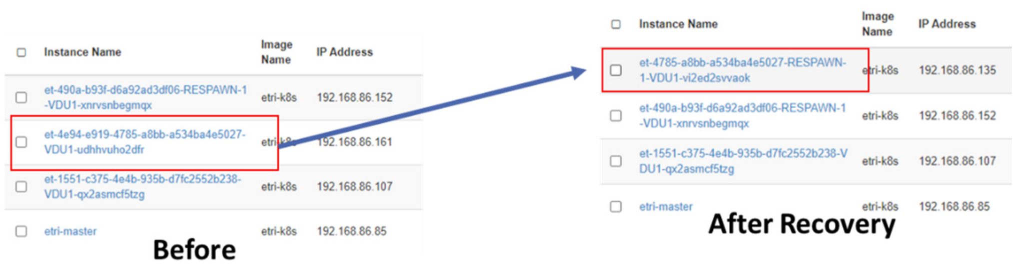


Figure 16. Recovery action based on Anomaly Score.

Figure 17 is the result of fault recovery facilitated by fault information. In this experiment, it is assumed that a fault has occurred as a result of one of the causes illustrated in Figure 15, and a specific server respawn was set to be used as a recovery operation to prevent faults.



Instance Name	Image Name	IP Address
et-490a-b93f-d6a92ad3df06-RESPAWN-1-VDU1-xnrvsnbegmqx	etri-k8s	192.168.86.152
et-4e94-e919-4785-a8bb-a534ba4e5027-VDU1-udhhvuho2dfr	etri-k8s	192.168.86.161
et-1551-c375-4e4b-935b-d7fc2552b238-VDU1-qx2asmcf5tzg	etri-k8s	192.168.86.107
etri-master	etri-k8s	192.168.86.85

Before

Instance Name	Image Name	IP Address
et-4785-a8bb-a534ba4e5027-RESPAWN-1-VDU1-vi2ed2svvaok	etri-k8s	192.168.86.135
et-490a-b93f-d6a92ad3df06-RESPAWN-1-VDU1-xnrvsnbegmqx	etri-k8s	192.168.86.152
et-1551-c375-4e4b-935b-d7fc2552b238-VDU1-qx2asmcf5tzg	etri-k8s	192.168.86.107
etri-master	etri-k8s	192.168.86.85

After Recovery

Figure 17. Recovery Result.

5. Conclusions

In this work, a machine learning-based cloud management system was proposed, developed and implemented. In addition, an architecture that can be used to identify and prevent faults in advance, by collecting data from the cloud infrastructure, was designed and implemented using open source cloud technologies. In order to check the change of numerous data, a method of calculating the anomaly score of the data collected in the cloud, using the self-supervised method, was utilized. Using the anomaly score, it was possible to quickly check the change situation of thousands of feature data, and based on this, the linkage structure with the failure recovery model was suggested. In addition, for the actual cause analysis, cause analysis was performed by adding a label to the anomaly score. The cause was analyzed using RFE, and the actual resource change pattern was confirmed based on the analysis result. If it was determined that there was an abnormal situation, the information was delivered to the recovery function. Through this, it was confirmed that the disadvantages of supervised learning based anomaly score can be supplemented, and fault can be prevented in advance. In a future study, we intend to conduct research on a model that can find the exact cause, through linkage, of additional cause analysis methods.

Contributions in this paper are as follows.

We designed a monitoring system for fault detection in a cloud environment and designed an architecture that links it to a deep learning-based fault detection architecture. For monitoring, a monitoring tool called Prometheus was used, and data were collected and learned. Subsequently, we designed an architecture that automatically labels the part to determine the exact cause of the faults and an architecture that can check the actual cause by using it. To identify the cause, the feature importance method was applied. Once a fault is determined, the interlocking structure with the cloud is designed so that the related restoration procedure can be automatically started. The entire designed architecture was implemented using open source, such as openstack and Kubernetes, and verified from the fault stage to the recovery stage.

6. Discussion

Cloud infrastructure has a complex architecture. With respect to this, the types of log data are also diverse, and the number of log data also changes as the number of virtual machines increases or decreases. Moreover, the average utilization rate of each resource varies according to the type of service and resources. In the cloud environment, cloud availability has been traditionally guaranteed through a monitoring system, but accurate fault detection is difficult due to the above-mentioned structural characteristics. To overcome this, fault detection techniques using supervised learning have been proposed, but the problem of labeling all data, securing data for fault learning, and detecting the causes of complex faults still remain. To overcome this, in this paper, we proposed a fault detection method using a self-supervised learning method that does not require labeling. In addition, for accurate fault detection, a cause analysis function that can find problems using abnormal data was also added. Finally, based on the detection result, a recovery function was also designed. All designed functions were integrated and implemented in the cloud environment and verified.

Author Contributions: All the authors contributed to the research and wrote the article. H.Y. proposed the idea, designed, and performed the evaluation. Y.K. suggested directions for the detailed designs and evaluation, as well as coordinating the research. All authors have read and agreed to the published version of the manuscript.

Funding: This work was partly supported by the Institute of Information & Communications Technology Planning & Evaluation (IITP) grant, funded by the Government of Korea (MSIT) (No.2020-0-00946, Development of Fast and Automatic Service recovery and Transition software in Hybrid Cloud Environment).

Institutional Review Board Statement: Not applicable.

Informed Consent Statement: Not applicable.

Data Availability Statement: Not applicable.

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

References

1. Guan, Q.; Fu, S. Adaptive Anomaly Identification by Exploring Metric Subspace in Cloud Computing Infrastructures. In Proceedings of the 2013 IEEE 32nd International Symposium on Reliable Distributed Systems, Braga, Portugal, 1–3 October 2013; pp. 205–214.
2. Pannu, H.S.; Liu, J.; Guan, Q.; Fu, S. AFD: Adaptive Failure Detection System for Cloud Computing Infrastructures. In Proceedings of the 2012 IEEE 31st International Performance Computing and Communications Conference (IPCCC), Austin, TX, USA, 1–3 December 2012; pp. 71–80.
3. Wang, C.; Talwar, V.; Schwan, K.; Ranganathan, P. Online Detection of Utility Cloud Anomalies Using Metric Distributions. In Proceedings of the 2010 IEEE Network Operations and Management Symposium—NOMS 2010, Osaka, Japan, 19–23 April 2010; pp. 96–103.
4. Wang, C.; Viswanathan, K.; Choudur, L.; Talwar, V.; Satterfield, W.; Schwan, K. Statistical Techniques for Online Anomaly Detection in Data Centers. In Proceedings of the 12th IFIP/IEEE International Symposium on Integrated Network Management (IM 2011) and Workshops, Dublin, Ireland, 23–27 May 2011; pp. 385–392.
5. Bianchini, R.; Fontoura, M.; Cortez, E.; Bonde, A.; Muzio, A.; Constantin, A.M.; Moscibroda, T.; Magalhaes, G.; Bablani, G.; Russinovich, M. Toward ML-Centric Cloud Platforms. *Commun. ACM* **2020**, *63*, 50–59. [CrossRef]
6. Bolivar, L.T.; Tselios, C.; Mellado Area, D.; Tsolis, G. On the Deployment of an Open-Source, 5G-Aware Evaluation Testbed. In Proceedings of the 2018 6th IEEE International Conference on Mobile Cloud Computing, Services, and Engineering (MobileCloud), Bamberg, Germany, 26 March 2018; pp. 51–58.
7. Salah, T.; Zemerly, M.J.; Yeun, C.Y.; Al-Qutayri, M.; Al-Hammadi, Y. Performance Comparison between Container-Based and VM-Based Services. In Proceedings of the 2017 20th Conference on Innovations in Clouds, Internet and Networks (ICIN), Paris, France, 26–29 March 2017; pp. 185–190.
8. Li, Z.; Kihl, M.; Lu, Q.; Andersson, J.A. Performance Overhead Comparison between Hypervisor and Container Based Virtualization. In Proceedings of the 2017 IEEE 31st International Conference on Advanced Information Networking and Applications (AINA), Taipei, Taiwan, 27–29 March 2017; pp. 955–962.
9. Kaur, K.; Dhand, T.; Kumar, N.; Zeadally, S. Container-as-a-Service at the edge: Trade-off between energy efficiency and service availability at fog nano data centers. *IEEE Wireless Commun.* **2017**, *24*, 48–56. [CrossRef]
10. Sauvanaud, C.; Lazri, K.; Kaaniche, M.; Kanoun, K. Anomaly Detection and Root Cause Localization in Virtual Network Functions. In Proceedings of the 2016 IEEE 27th International Symposium on Software Reliability Engineering (ISSRE), Ottawa, ON, Canada, 23–27 October 2016; pp. 196–206.
11. Liu, J.; Chen, S.; Zhou, Z.; Wu, T. An Anomaly Detection Algorithm of Cloud Platform Based on Self-Organizing Maps. *Math. Probl. Eng.* **2016**, *2016*, 3570305. [CrossRef]
12. Cotroneo, D.; Natella, R.; Rosiello, S. A Fault Correlation Approach to Detect Performance Anomalies in Virtual Network Function Chains. In Proceedings of the 2017 IEEE 28th International Symposium on Software Reliability Engineering (ISSRE), Toulouse, France, 23–26 October 2017; pp. 90–100.
13. Soualhia, M.; Fu, C.; Khomh, F. Infrastructure Fault Detection and Prediction in Edge Cloud Environments. In Proceedings of the 4th ACM/IEEE Symposium on Edge Computing (SEC '19). Association for Computing Machinery, New York, NY, USA, 7–9 November 2019; pp. 222–235.
14. Wang, B.; Hua, Q.; Zhang, H.; Tan, X.; Nan, Y.; Chen, R.; Shu, X. Research on anomaly detection and real-time reliability evaluation with the log of cloud platform. *Alex. Eng. J.* **2020**, *61*, 7183–7193. [CrossRef]
15. El-Shamy, A.M.; El-Fishawy, N.A.; Attiya, G.; Mohamed, M.A. Anomaly Detection and Bottleneck Identification of The Distributed Application in Cloud Data Center using Software–Defined Networking. *Egypt. Inform. J.* **2021**, *22*, 417–432. [CrossRef]
16. Garg, S.; Kaur, K.; Kumar, N.; Rodrigues, J.J.P.C. Hybrid Deep-Learning-Based Anomaly Detection Scheme for Suspicious Flow Detection in SDN: A Social Multimedia Perspective. *IEEE Trans. Multimed.* **2019**, *21*, 566–578. [CrossRef]
17. He, Z.; Lee, R.B. CloudShield: Real-time Anomaly Detection in the Cloud. *arXiv* **2021**, arXiv:2108.08977.
18. Vu, D.D.; Vu, X.T.; Kim, Y. Deep Learning-Based Fault Prediction in Cloud System. In Proceedings of the 2021 International Conference on Information and Communication Technology Convergence (ICTC), Jeju Island, Republic of Korea, 20–22 October 2021; pp. 1826–1829.
19. Gao, J.; Wang, H.; Shen, H. Task Failure Prediction in Cloud Data Centers Using Deep Learning. In Proceedings of the 2019 IEEE International Conference on Big Data (Big Data), Los Angeles, CA, USA, 9–12 December 2019; pp. 1111–1116.
20. Openstack. Available online: https://wiki.openstack.org/wiki (accessed on 14 March 2021).
21. Kubernetes. Available online: https://kubernetes.io/docs/home/ (accessed on 14 March 2021).
22. Kourtis, M.; Mcgrath, M.J.; Gardikis, G.; Xilouris, G.; Riccobene, V.; Rapadimitriou, P.; Trouva, E.; Liberati, F.; Trubian, M.; Batalle, J.; et al. T-NOVA: An Open-Source MANO Stack for NFV Infrastructures. *IEEE Trans. Netw. Serv. Manag.* **2017**, *14*, 586–602. [CrossRef]
23. Li, B. *Anomaly Detection in Streaming Data using Autoencoders*; Hannover University: Hannover, Germany, 2018.
24. Malhotra, P.; Ramakrishnan, A.; Anand, G.; Vig, L.; Agarwal, P.; Shroff, G. Lstm-based encoder-decoder for multi-sensor anomaly detection. *arXiv* **2016**, arXiv:1607.00148.
25. Russo, S.; Disch, A.; Blumensaat, F.; Villez, K. Anomaly Detection using Deep Autoencoders for in-situ Wastewater Systems Monitoring Data. *arXiv* **2020**, arXiv:2002.03843.

26. Ahmad, S.; Purdy, S. *Real-Time Anomaly Detection for Streaming Analytics*; Numenta: Redwood City, CA, USA, 2016.
27. PromQL. Available online: https://prometheus.io/docs/prometheus/latest/querying/basics/ (accessed on 14 March 2021).
28. Stress-ng. Available online: http://kernel.ubuntu.com/~cking/stress-ng/ (accessed on 14 March 2021).
29. Iperf3. Available online: https://iperf.fr/ (accessed on 14 March 2021).
30. Pytorch. Available online: https://pytorch.org/ (accessed on 14 March 2021).
31. Prometheus. Available online: https://prometheus.io/ (accessed on 14 March 2021).
32. Gunasegaran, T.; Cheah, Y. Evolutionary Cross Validation. In Proceedings of the 2017 8th International Conference on Information Technology (ICIT), Amman, Jordan, 17–18 May 2017; pp. 89–95.
33. Hajian-Tilaki, K. Receiver Operating Characteristic (ROC) Curve Analysis for Medical Diagnostic Test Evaluation. *Caspian J Int. Med.* **2013**, *4*, 627–635.
34. Ke, G.; Meng, Q.; Finley, T.; Wang, T.; Chen, W.; Ma, W.; Ye, Q.; Liu, T.-Y. Lightgbm: A highly efficient gradient boosting decision tree. *Proc. Adv. Neural Inf. Process. Syst.* **2017**, *30*, 3146–3154.