Master's Thesis

Virtual Machine Failure Prediction using Log Analysis

Sukhyun Nam (남 석 현)

Department of Computer Science and Engineering

Pohang University of Science and Technology

2021

로그 분석 기반 가상 머신 고장 예측

Virtual Machine Failure Prediction using Log Analysis

Virtual Machine Failure Prediction using Log Analysis

by

Sukhyun Nam
Department of Computer Science and Engineering
Pohang University of Science and Technology

A thesis submitted to the faculty of the Pohang University of Science and Technology in partial fulfillment of the requirements for the degree of Master of Science in the Computer Science and Engineering

Pohang, Korea
06. 22. 2021
Approved by
James Won-Ki Hong (Signature)
Academic advisor

Virtual Machine Failure Prediction using Log Analysis

Sukhyun Nam

The undersigned have examined this thesis and hereby certify that it is worthy of acceptance for a master's degree from POSTECH

06. 22. 2021

Committee Chair	James Won-Ki Hong	(Seal)
Member	Jae-Hyoung Yoo	(Seal)
Member	Inseok Hwang	(Seal)

MCSE 남 석 현 Sukhyun Nam
20192016
Virtual Machine Failure Prediction using Log Analysis.
로그 분석 기반 가상 머신 고장 예측.
Department of Computer Science and Engineering , 2021, 33p
Advisors : Prof. James Won-Ki Hong, Prof. Jae-Hyoung Yoo.
Text in English.

ABSTRACT

In this study, we propose a machine learning model that predicts failures by analyzing logs before failures occur in virtual machines (VMs) used in network function virtualization (NFV) environments. The proposed model utilizes a convolutional neural network (CNN) and includes pre-processing and pre-failure tagging techniques. We collected log data from VMs built on OpenStack to validate the proposed model. We classified failures based on early fault messages and built a CNN model to predict VM failures with fault messages. The experimental results showed that the proposed model can predict failures within a 5-min period with an F1-score of 0.67. The proposed model could be used for VM proactive live migration to avoid service degradation and interruptions caused by failures.

Contents

List of Tables

List of Figures

I. Introduction

Modern networks are becoming larger in size and more complex in structure. As the public's consumption of Internet content increases, the demand for high-throughput services is also increasing. In traditional network environments, service providers install expensive hardware-based middleboxes inside the network to provide various services. However, the installation and operation costs increase as the network size increases; therefore, network function virtualization (NFV) technology has been proposed as an alternative. NFV is a technology that operates network functions such as firewalls and an intrusion detection systems (IDS) through virtual machines (VMs) operating on virtualized servers without installing them on middleboxes (Fig. 1.1). In a traditional network, if network requirements are changed, new middleboxes have to be installed, whereas in an NFV environment, network administrators can replace the hardware with virtualized network functions (VNFs), which are cheaper and easier to manage. By replacing expensive middleboxes with VNFs, the cloud system can respond quickly and flexibly to fluidly changing service requests [1] and use the resources more efficiently [2].

The advent of NFV technology has further complicated existing complex software-defined networking (SDN) based structures. Although SDN and NFV have reduced CAPEX/OPEX, such complex virtual structures have made it more difficult to monitor and take action on virtual networks and server failures. To address this problem, many studies on anomaly detection in SDN/NFV environments are ongoing [3, 4]; however, there remains a lack of research on technologies that proactively predict failures in servers, VMs, and VNFs to allow action to be taken before a failure occurs.

The failure prediction problem in NFV is challenging for several reasons. First, owing to the complexity of large-scale network systems, failures can be caused by many heterogeneous software and hardware faults. Second, the failure data are usu-

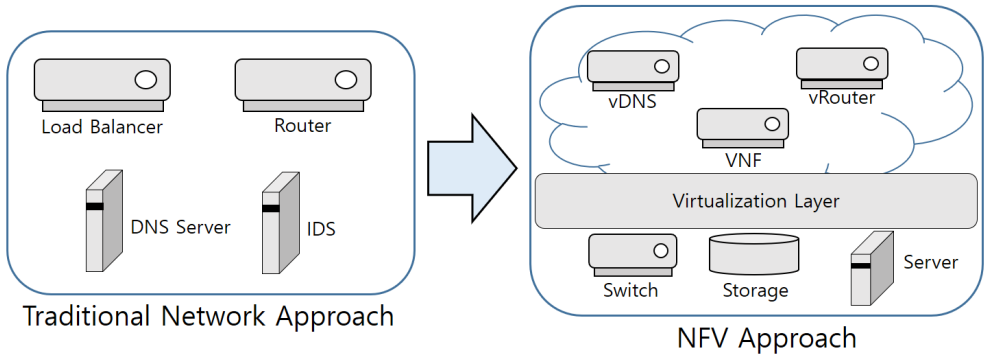

Figure 1.1: Traditional network structure and NFV

ally highly unbalanced. For example, Microsoft cloud services reported that each day, less than 0.1% of nodes encounter failures [5]. Unbalanced data lead to a poor performance of the prediction model. Third, it is difficult to determine the symptoms of failures from a large number of data indicating the state of the network equipment. In the case of log data, for example, modern cloud systems produce approximately 30-50 gigabytes (approximately 120-200 million lines) of logs per hour [6]. However, most failures have associated symptoms, and recent advances in machine learning technologies have made it possible to analyze huge numbers of complicated data. Recent studies conducted in various network management fields have also achieved a high performance using machine learning [7, 8].

A Mandelbug [9] is a type of fault with a complex activation and propagation; thus, it is difficult to reproduce and incurs a high possibility of a time lag between a fault activation and a failure occurrence. Based on an empirical study on bug reports in a Linux kernel [10], network faults are more likely to be of a Mandelbug type because networking is regarded as a basic and core function from an operating system aspect, and thus its interaction is more complex and tight. The average time required to fix a Mandelbug is longer than that of regular bugs [10]. Therefore, faults and failures in networks and servers are dangerous and can cause serious losses to service operators. For example, faults in clouds deployed on OpenStack can take hours or even days to fix

[11]. However, most failures have previous faults or errors before they occur. Failures based on fault events can be predicted if failures have former faults. In the case of VNF, if we can predict a failure in advance, we can migrate the VNF to another server before a failure occurs to minimize the service quality degradation.

Faults and errors in the computer equipment can be found in the log. Most server and network equipment provide real-time log outputs (e.g., syslog). Although studies are underway to harness logs for network management [12, 13, 14, 15, 16], because the software structure of servers and networks has become more complex, the number of logs has increased proportionally. In addition, because modern systems are vast and complex, each log of a subsystem is usually made by a single developer who might have an incomplete understanding of the overall system behaviors [13]. Most open-source based systems are developed by hundreds of developers, and thus the logs are not systematically generated. The automatic analysis of log data therefore remains a challenge.

In a log analysis, sentence classification techniques using natural language processing (NLP) techniques are applied. Sentence classification is the field of analyzing textual documents such as movie reviews to determine likes and dislikes of movies, or to classify documents according to predefined criteria such as spam mail classification. Unlike other NLP fields where recurrent neural networks (RNNs) are strong because of the importance of word order, convolutional neural networks (CNNs) are mainly used to analyze how each word affects the classification results [17, 18].

In this thesis, we propose a CNN-based VM failure prediction model that uses logs extracted from each VM as input values. In this study, we define a failure as a state in which a VM in the network fails to perform network functions. The model predicts the failure with former fault messages. The model includes additional techniques for a log analysis, such as log-based word embedding and pre-failure tagging techniques.

To validate the feasibility of the proposed model, we trained the model with logs collected from VMs for two weeks in an NFV environment. The experimental results showed that the F1-score was 0.67, which predicted a failure within a 5-min period.

We provide the following contributions:

1. We construct an NFV environment with OpenStack to induce failures and classify failures in which faults exist based on logs.

2. We use a CNN to generate a model that receives the log of the VM as an input and predicts failure within a few minutes, and evaluate the model.

3. To improve the performance of the CNN models, we propose a pre-failure tagging method.

II. Related Work

2.1 Faults and Failures

Because computer systems are extremely complex, the causes of failures also vary widely, and thus the ways to cope with them are different. Understanding the type of faults that cause a failure can provide valuable insight into system maintenance; therefore, research is being conducted on the characteristics and classification of faults. In general, faults are classified as Bohrbug and Mandelbug faults [9]. Whereas Bohrbug faults can be easily isolated and reproduced, Mandelbug is an opposite terminology whose activation and propagation appear non-deterministic and is complex. In addition, Mandelbug faults have two possible cases that are not mutually exclusive [9]: First, such a fault causes a failure that is influenced by other elements (e.g., operating system or hardware) of the software system. Second, the complexity of the error propagation results in a time lag between fault activation and final failure occurrences. In the Mandelbug sub-classification system proposed by [19], LAG and ARB correspond to the second case, which has a delay in failure occurrence.

- ARB (Aging Related Bug): This is a kind of bug that can cause an increasing failure rate and/or degraded performance, known as software aging, representing a situation in which the error state is generated slowly owing to an overload, such as continuous memory leaks or an increase in the total system runtime.
- LAG: This is a type of non-aging-related Mandelbug (NAM); however, there exists a time **lag** between the activation of the bug and the occurrence of its failure.

In either case, the error conditions do not immediately lead to failures. These features provide the possibility of predicting a failure through faults. In this study,

we classified ARB and LAG from the failures we collected, based on error logs, to determine whether faults existed before a failure. We then utilize the failures classified as LAG and ARB for CNN model learning.

2.2 Sentence Classification

Sentence classification is a field in an NLP. We introduce techniques related to sentence classification because we applied these techniques to building a CNN-based log-analysis model.

2.2.1 Word Embedding

Word embedding [20] is a technique that expresses a word as a dense vector while preserving the characteristics of that word. Fig. 2.1 shows the concept of word embedding. Before word embedding was proposed, a word was expressed as a one-hot vector (sparse representation). However, there was a problem with these expressions, in which the number of words increases in a corpus, and the dimensions of the vectors become infinitely larger. If the corpus has 10,000 words, the number of dimensions of each word vector must also be 10,000. In addition, the one-hot vector causes a huge waste of space. To remedy these problems, word embedding techniques are intended to represent words as vectors with real numbers. Word embeddings set the dimensions of the vectors of all words to the values set by the user, and learn as the vector expresses the characteristics of each word.

There are currently a variety of word embedding methodologies, of which word2vec [21] is the most commonly used. Word2vec utilizes two methods: continuous bag-of-words (CBOW) and Skip-gram. CBOW predicts words in the middle of a sentence (center word) with words around them (context words). By contrast, Skip-gram is a method of predicting context words using the center word. For example, in the Skip-gram approach, a window of a user-specified size moves a sentence, learning a shallow neural network with an input as a one-hot vector of the center word, and an output as

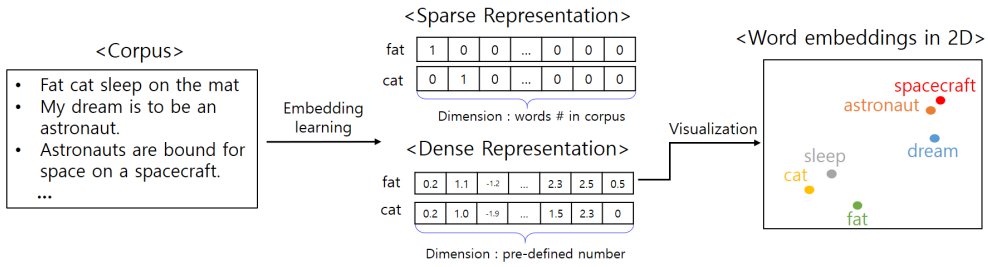

Figure 2.1: Concept of word embedding

one-hot vectors of the context words. At this point, the weight value changes according to each word, and the final weight value becomes the word vector for each word. This learning technique makes word vectors of words that are close in a sentence to be close to each other in the vector space.

2.2.2 CNN-based Sentence Classification

A CNN is a type of artificial neural network that is specialized for extracting features without losing information from large amounts of data. CNNs contain multiple convolution layers and pooling layers, their operation is simple, and the number of their parameters is small. The convolution layer extracts features by conducting convolutional operations with filters (kernels) on the data. The pooling layer reduces the size of the output data in the convolution layer or highlights specific data by leaving only the maximum value of a matrix (max pooling), or by converting it into an average value (average pooling). CNNs used in computer vision fields and others use multiple layers of convolution and pooling layers to extract features from highly complex data; however, in a sentence classification problem, the CNN model contains only one convolution layer and a pooling layer.

Kim [17] proposed the use of a CNN model in a sentence classification problem. This study used word embedding vectors [20] as input features for the CNN. The author generated sentence vectors through a concatenation process with the generated

word embedding vectors. The generated sentence vector was input through a convolution layer. The filters used in the convolution layers used the dimensions of the product of filter size h and size of the embedding vectors k, that is, hk. Because the length of the input sentence varies, the number of dimensions of the vector generated at this point is not constant. Thus, the max-pooling layer takes the maximum value for each vector generated from the filter. The idea is to capture the most important features of each feature map. Finally, feature values are generated with the number of filters and are passed to a fully connected softmax layer whose output is the probability distribution over the labels. Although the performance of the proposed CNN model was not significantly superior to that of the other algorithms (e.g., RNN-based and SVM algorithms), it showed a slightly better performance than the other models despite being built using the simplest form of a CNN, which indicates that a CNN fits the sentence classification problems.

2.3 Anomaly Detection & Prediction

Studies on the detection of abnormal situations in SDN/NFV environments have been one of the most studied topics in the network field over the past few years; however, there has been little research on prediction topic yet.

Hong *et al.* [3] proposed a machine learning-based VNF anomaly detection method using instance information and resource usage of VNFs. The authors built an NFV infrastructure (NFVI) with an OpenStack environment and operates various services on it. They collected VNFs and network status data using a fault injection process because anomalies do not occur frequently in the network. They generated faults by: 1) generating the abnormal states of a VM, and 2) generating a heavy overload in network traffic. Their model showed an F1-score of over 95% for anomaly detection and 93% for root-cause localization.

Du *et al.* [15] proposed a contextual anomaly detection model using a deep neural network (DNN) to learn log patterns from a normal execution and detect anomalies

when log patterns deviate from the model. They parsed each log entry into a log key and parameter value vector and trained two parallel models as multiclass classifiers with a log key and parameter value vector. Their model showed an F1-score of 0.98 with the OpenStack log data set.

Lin *et al.* [5] proposed a node failure prediction model for cloud service systems. They collected temporal features (e.g., performance counters and resource usage) and spatial features (for example, the rack location, load balance group, and update domain) from a production cloud service system. They proposed an ensemble of machine learning models to combine heterogeneous data from diverse sources. Their model predicts a node failure before 6 h with an average recall of 0.63 and an F1-score of 0.75.

Liang *et al.* [16] collected event logs from an IBM supercomputer and proposed a model that predicts whether fatal events will occur in the next interval (prediction window) based on the events in the observation period (input window). They did not apply NLP methods in a log analysis, and simply extracted event-related features from logs (e.g., warning events occurred during the current window). The results show that their proposed model could predict a failure with an F1-score of 70% for a 12-h window size.

Ji *et al.*[14] proposed a CNN model that collects log data from wireless communication systems in a simulation environment to predict whether log data after a constant gap contained failure messages. In this study, the authors used a pre-processing technique that removed both symbols and numbers from the log data. The CNN model is based on the model in [17]. This work was compared to models using long short-term memory (LSTM) and gated recurrent units (GRU) for a performance verification of the proposed CNN model, and the results showed that the CNN model achieved a slightly higher performance than the LSTM and GRU model. In addition, they trained and tested the model according to the gap and input window sizes. The learning results showed an accuracy of approximately 0.75 when the gap was 2000, and an accuracy of approximately 0.57 when the gap was 5000. For the failure prediction and detection

problems, the performance should be compared through figures that can infer false-negative rates, but they only specify the accuracy. Furthermore, the time difference between the failure occurrence and input window is not constant because the unit of the gap is defined as the number of log messages. In particular, because more log messages are generated when faults occur, the time difference with the prediction target is further reduced, resulting in a more urgent prediction.

Thus, in our research, we used time (minutes) as a unit of the gap to stabilize the time difference from the prediction target. We also used a similar model with the CNN model from [17], but we added pre-failure tagging steps to the data tagging stage to generate more suitable data for a log analysis.

III. Design

This chapter describes the entire design of the proposed failure prediction model and provides a detailed description of each element.

3.1 Design of Failure Prediction Model

We propose a model that uses log data to predict and alert failure risks in a VM. The proposed model uses a constant time-based gap until the prediction point is reached. Fig. 3.1 shows an overview of the deployment process of the proposed CNN-based VM failure prediction model.

The CNN model learns to calculate the probability that a failure will occur after *gap* minutes based on the *input window size* of the input logs. To train our model, we need to generate log data as the input and failure state data as the output. In the data generation step, we collected logs and state information from each VM. The log data go through a pre-processing, are converted into sentence embedding, and then delivered to the CNN model. Failure history data are also delivered to the CNN model.

Fig. 3.2 shows a detailed example of the input and output of the model. Two parallel sliding windows are used. Each window moves on the log data and the failure history data. We extracted the input sequence in the first sliding window and extracted the output label from the second sliding window. The input window has an *input window size* of minutes. The input and output windows have a constant time interval, defined as *gap* minutes. The windows slide for 1 min and generate input logs and output labels.

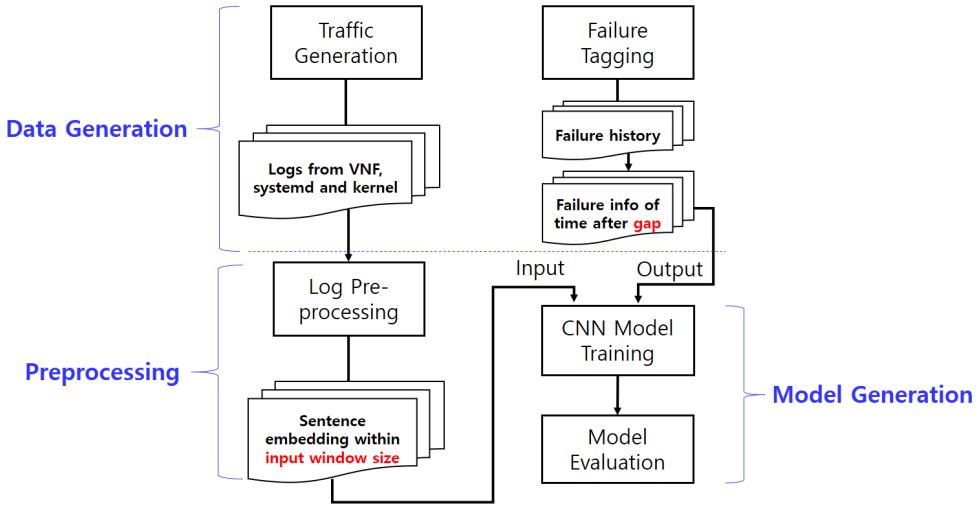

Figure 3.1: Overview of predicting model deployment process

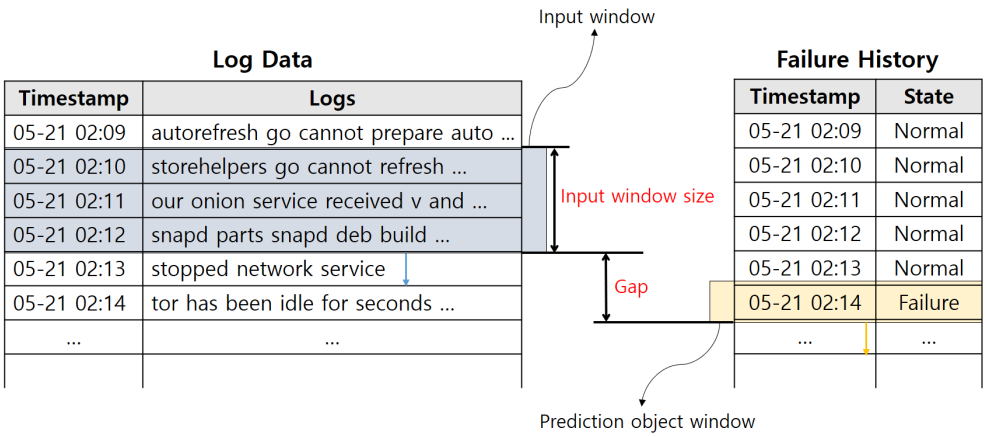

Figure 3.2: Sliding window in log data and failure history

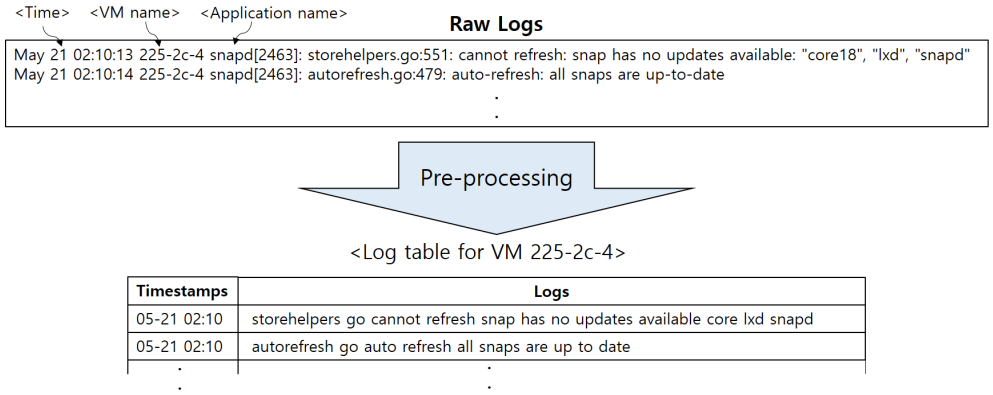

Figure 3.3: Log pre-processing example

3.2 Input Data

We used sliding windows with an *input window size* of minutes to obtain the input sequence from the logs. We removed the numbers and replaced the symbols with spaces, removing the VM information, time, and application names from the logs. Fig. 3.3 illustrates an example of logs that have been pre-processed. In addition, if there were duplicate sentences in the window, they were removed, leaving only one sentence.

The generated log corpus was converted into log embedding through word embedding before being entering the CNN model. Because word embedding learns similarity through words that are together within sentences during the learning phase, public word embeddings are unsuitable for a log analysis. Therefore, we applied a word embedding generated using our log corpus data. For word embedding learning, we used Google's open-source project word2vec [21] and collected a log corpus for 1-month period from six VMs and servers in our testbed. We set the minimum count to learn only the words that appeared more than once in the daily log for each VM. We used Skip-gram as an embedding algorithm, and we set the vector size to 100 and window size to 5. The generated word embedding contained 265,452 words. Based

on the results, for example, the closest words to ”err” were ”over”, ”dropped”, ”rx”, ”crc”, ”tx”, ”collison” and ”miss.”

3.3 Convolutional Neural Network

We built a CNN model based on the CNN model from [17]. Fig. 3.4 shows the CNN model and the inputs and outputs used in our work. The generated word embedding goes through a convolutional layer, which consists of several types of filters. We set the number of filter types to $\lfloor \textit{input window size}/2 \rfloor + 1$ because the larger the input value, the more necessary the filter is, and the size of the filter starts at 3 and grows by increments of 1. For example, if the *input window size* is 5, each filter has dimensions of 300, 400, and 500 because word embedding has dimensions of 100. The pooling layer is used to leave only the most important value from each vector generated by the filters. According to [22], using the max-pooling layer for a CNN in the sentence classification problem showed the highest performance; therefore, we also used the max-pooling layer. The max-pooling layer produces a vector with the same number of dimensions as the number of filters. We then put the vector into the fully-connected layer and dropout layer. A dropout layer was used to prevent an overfitting. In addition, we used Sigmoid as an activation function in the fully-connected layer to calculate the probability of a failure occurring as a value between zero and 1.

Our training data had output values of between zero and 1. When learning probability distributions, the model typically uses the cross-entropy loss function or the Kullback–Leibler divergence (KL divergence) loss function. However, the binary cross-entropy loss function works only when the prediction object is zero or 1, and the KL divergence loss always returns the zero if the true value is zero. Therefore, we utilize the following KL divergence-based loss function: y_{true} represents the tagged (true) values, and y_{pred} represents the output of the CNN (prediction result). We also applied class weights to handle unblanched data, which will be explained later.

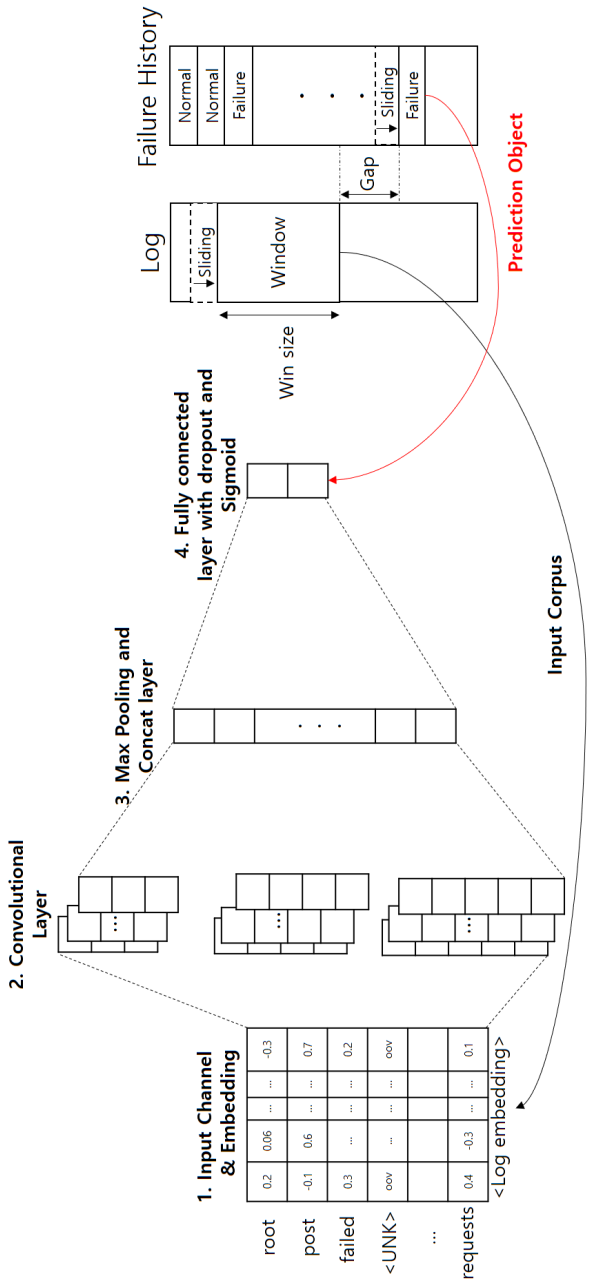

Figure 3.4: CNN model design

$$loss = y_{true} \times log\left(\frac{y_{true}}{y_{pred}}\right) \times ClassWeight_1 + (1 - y_{true}) \times log\left(\frac{1-y_{true}}{1-y_{pred}}\right) \times ClassWeight_0$$

3.4 Output Tagging

We need to check the status of the VMs, and thus we sent a ping message from the other server (state checker) to each VM every minute to check the status. We decided to tag the VM as a failure when it refused to ping during a 1 min period. Fig. 3.5 shows the deterministic finite automata (DFA) representing the tagging scheme. In addition, when multiple VMs were in a failure state at the same time, it was excluded from the VM failure tag, judging that there was a problem with the server, and not with the VM.

Learning the CNN model requires a label for each input window. Similar to the general classification problem, we tagged each input window as 1 if it was related to a failure and zero if the state was normal. We tagged the data based on the failure history, which is the data recorded every minute whether each VM failed or was in a normal state. Each input window was tagged to determine whether a failure occurred after *gap* minutes according to the failure history.

We could further increase the performance through the a pre-failure tagging method, which tags the states before the failure occurred as a pre-failure rather than as normal. Fig. 3.6 shows the difference between regular tagging and pre-failure tagging. In this

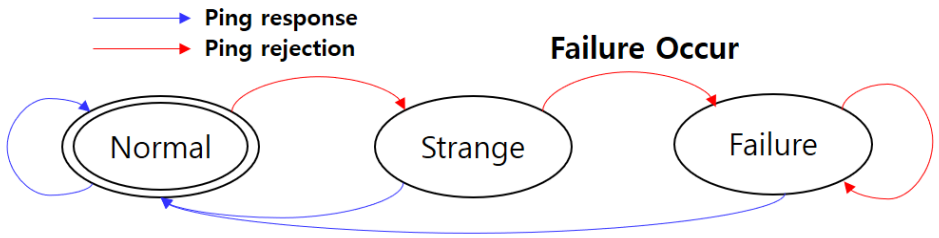

Figure 3.5: VM state tagging DFA

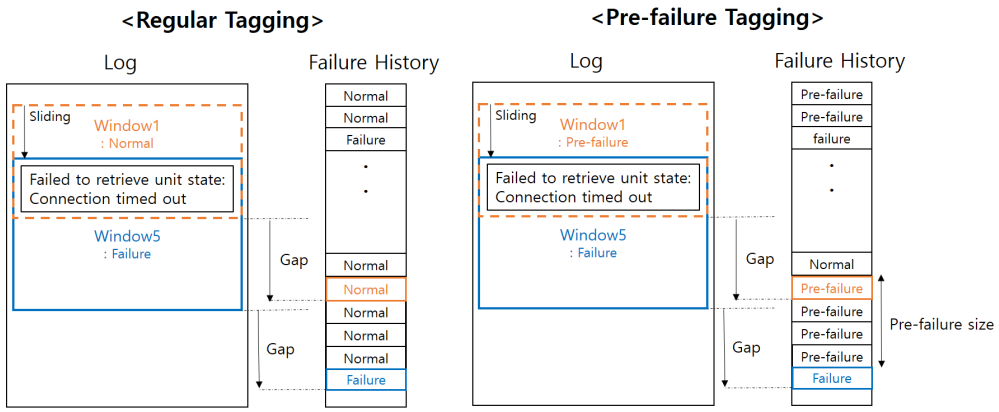

Figure 3.6: Regular tagging and pre-failure tagging

illustration, with regular tagging, Windows 1 through 4 are tagged as zero because the state is normal after gap minutes, and Window 5 is tagged as 1. However, error messages associated with the failure (in this case, "Failed to retrieve unit state: Connection timed out") are included in all windows from Window1 through Window5. Owing to the nature of a CNN, which extracts only important features regardless of the order of the words, error messages are highly likely to not be recognized as the cause of the failure because Window 1 through 4 are learned to output a value of zero. Therefore, we tagged the normal states as pre-failure states during the *pre-failure size* minutes just before the occurrence of a failure. In addition, we tagged windows with a pre-failure state as a *pre-failure value*, which is a value between zero and 1. Therefore, in our model, the CNN can learn about the error message even if there is no failure after the *gap*. We conducted experiments to determine which values were appropriate for the *pre-failure size* and *pre-failure value*.

With the pre-failure tagging method, windows near a failure will output values that are close to *pre-failure values* but must be tagged as normal. Therefore, we set the threshold of failure as a *pre-failure value*+0.05 to prevent the output of the pre-failure window from exceeding the threshold.

IV. Experiment and Evaluation

In this section, we describe the experiments conducted to verify the suitability of the proposed model along with the results.

4.1 Experimental Setup

We used the NFVI built using OpenStack as a testbed for collecting the data. Fig. 4.1 shows the detailed architecture of our testbed. This NFVI environment operates a variety of VNFs (e.g., IDS and firewall) to provide network services. Multiple servers work as compute nodes, and each node contains VMs to run the VNF and serve as clients and servers. We installed six different VNFs on each VM to output various logs and create different situations. The list of VNFs and VMs is presented in Table 4.1. We changed the specifications of each VM in response to the requirements of each VNF.

We generated multiple client–VNF-server chains that utilize each VNF as a single service, allowing each VNF to handle the traffic. We used Apache as the web server. Each VM passes logs from the VNF, system daemon, and kernel to the monitoring node using rsyslog [23]. The state checker VM checks the states of the VMs with a periodic ping and sends the state information of each VM to the monitoring node.

The monitoring node runs a rsyslog server and collects the failure history data. The monitoring node extracts the log of the *input window size* from the collected log data using a timestamp and pre-processes them. Finally, the monitoring node passes the pre-processed log data and the output label after the *gap* to the AI node.

An AI node contains pre-trained word embedding and a CNN model. The CNN learns the probability that a failure has occurred after a *gap* of minutes with the delivered data. In the future, when we develop our research and apply it to a live migration,

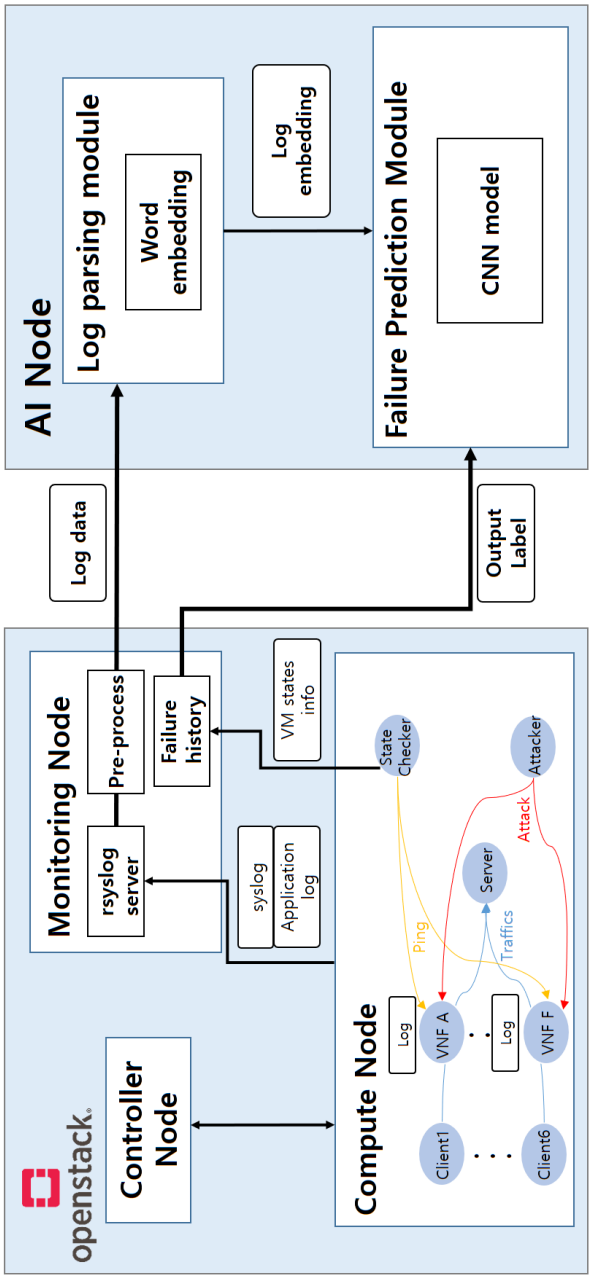

Figure 4.1: Testbed implementation

Table 4.1: VNF list

VNF	Type	VM specification
Suricata	IDS	CPU 4 core, Memory 4G
Haproxy	Load Balancer	CPU 4 core, Memory 4G
IPtables	Firewall	CPU 2 core, Memory 1G
ntopng	Network Monitoring	CPU 2 core, Memory 1G
nDPI	DPI	CPU 2 core, Memory 1G
Snort	IDS	CPU 2 core, Memory 4G

the AI node will pass the alert to the control node when the failure probability crosses the predefined threshold. The control node can then move the VNF to another server to prevent a failure before the it occurs through a VM live migration.

4.2 Data Collection

A failure does not occur easily in situations in which VMs are ordinarily operated. Therefore, we overloaded the VMs in three ways to generate as many failures as possible (Fig. 4.2).

- Resource overload: We overloaded the CPU and memory for each VM using Stress-ng [24], a resource overload tool. We continuously increased the CPU usuage and memory usage. The overload started at 50% and increased by 5% every 5 min, and when a failure occurred, the overload figures were reset.
- Traffic overload: The client VM sends traffic requests to the server using Apache Bench [25], a server performance checking tool, and it continuously increases the number of requests and increases the number of concurrent connections every 30 min. If a failure occurs, the overload figures are reset.

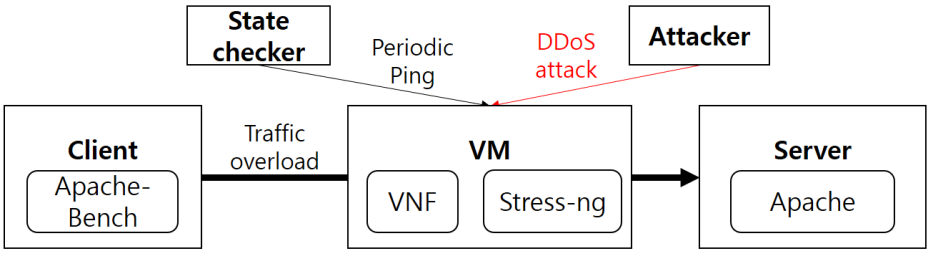

Figure 4.2: Data generating method with overloads

- External attack: The attacker server generates a DDoS attack on a randomly chosen VM with hping3 [26], a security tool. It transmits more than 100,000 packets at intervals of below 6μs. The transmission packet intervals and attack times were randomly selected at each time of the attack.

As a result, we collected 44 failures within a month of the experiment. Thirteen of them occurred simultaneously in the VMs, and thus they were determined to be from a server failure. Based on the log, we select the failures where the former fault logs exist. Four of the 31 VM failures that did not have an error log before the failure were determined. The remaining 27 failures were considered to have early faults associated with the failure and were distinguished by ARB and LAG according to the criteria provided in [19].

In the case of ARB, similar fault logs were repeated continuously, and repeated logs were mainly regarding messages indicating that processes or networks were unresponsive or took too long. In the case of LAG, there was an error message that did not normally occur, and in general, other system daemons were restarted after the message occurred. Furthermore, error logs mainly occurred from the kernel and were related to hardware or system faults. The following are examples of log messages from the ARBs and LAGs observed before a failure.

- ARB

  - task_delay_info Worker processing SEQNUM is taking a long time
  - Failed to retrieve unit state: Connection timed out

- LAG
  - blk_update_request: I/O error, dev vda, sector op READ flags phys_seg prio class
  - fail to add MMCONFIG information, can't access extended PCI configuration space under this bridge

ARBs occurred much more frequently, with 6 of the 27 failures being LAGs and 21 being ARBs. If the time difference between the fault and the failure was too long, it was excluded because the fault message did not enter the input window at the time of training, and thus five failures with a time difference of more than 30 min were excluded.

4.3 CNN Learning

The last generated data included 22 failures. However, in experimental environments with an *input window size* of 5 min and a *gap* of 5 min, more than 1,500 windows were created per day (we did not create an input window when the log did not exist). Because the data were excessively unbalanced, we applied the oversampling by 2 times to the failure data and the undersampling by 60 times to normal data. In addition, we applied the class weight as the reciprocal number of each class (normal/failure) in the data to the loss function.

We implemented our CNN model using Keras, and we used the L2 regularizer and the ReLU activation function and the Adam optimizer. All codes that we used were opened in [27]. The specifications of the computational machine include an Intel Xeon Silver 4215@2.50 GHz processor, 64 GB of RAM, and an NVIDIA RTX 50000 GPU running a 64-bit Ubuntu 18.04 operating system and the NVIDIA CUDA Depp Neural Network library (cdDNN).

We experimented to determine the appropriate value for the *pre-failure size*, *pre-failure value*, *input window size*, and *gap*. In the case of the *gap*, we measured the VM live migration time to capture the lower bound. According to [28], the live migration in OpenStack includes nine steps, and the original VM provides service until the 6th step, stop-and-copy, which is the step during which the VM is paused. Therefore, we determined that the *gap* should be longer than the time taken to reach the 6th step. We tested how long it would take to conduct the five steps. As a result, it took an average of 45 s before the stop-and-copy phase, based on VMs with a size of 5 GB on OpenStack. Therefore, we decided that even if the *gap* was 1 min, a migration could occur before a failure occurred under normal circumstances.

In NLP, words that are not in the dictionary are called out-of-vocabulary (OOV). In general, for OOV, the model does not apply word embeddings, and instead uses randomly generated vectors or pre-defined OOV vectors. In the case of log data, most of the OOVs were an application internal ID or process ID such as ”UOixfW” and ”xdc” (word after pre-processing). However, in the case of LAG errors, they generated logs that had never been observed before, and thus included OOV in a input window with a high probability. In fact, the error log of one of the failures we observed in our experiment was ”sysrq: Resetting”, and both words were OOV with our pre-learned word embedding. It is more important to catch logs related to faults than to delete unimportant IDs, and thus we tagged OOV with a random vector of 100 dimensions.

4.4 Results

We collected 35,370 data, and using an undersampling, we applied only 589 data. We randomly separated the data into training data and test data at a 8:2 ratio. In addition, 20% of them were used as a validation set, and thus the model was trained until the loss value for the validation set remained unchanged.

First, we tested the appropriate values for the *pre-failure size* and *pre-failure value*. At this time, the *input window size* and *gap* were set to 5 min. First, we exper-

Table 4.2: Pre-failure size and pre-failure value test

Size (min) \ Value	0.5	0.65	0.8
3	Acc: 0.98, Rec: 0.75, F1: 0.60	**Acc: 0.95, Rec: 1.00, F1: 0.67**	Acc: 0.97, Rec: 0.60, F1: 0.55
5	Acc: 0.97, Rec: 0.57, F1: 0.57	Acc: 0.91, Rec: 0.33, F1: 0.29	Acc: 0.96, Rec: 0.33, F1: 0.40
10	Acc: 0.86, Rec: 0.75, F1: 0.44	Acc: 0.90, Rec: 0.40, F1: 0.25	Acc: 0.93, Rec: 0.50, F1: 0.36

imented without pre-failure tagging as a control group. Without pre-failure tagging, the CNN showed an accuracy of 0.95 and an F1-score of 0.25, which was extremely low. We experimented by changing the *pre-failure size* to 3, 5, and 10 min and the *pre-failure value* to 0.5, 0.65, and 0.8. As a result, the accuracy was generally similar and the F1-score varied, but was much higher than the result of the test without pre-failure tagging. Table 4.2 shows the results. The highest performance was an F1-score of 0.67 when the pre-failure tagging 0.65 for three minutes. In general, the performance increased when the *pre-failure size* decreases.

We experimented by changing the *gap* to 1, 3, 5 and 10 min, and the *input window size* to 5, 10, and 20 min. At this time, we utilized a *pre-failure size* of 3 min, and a *pre-failure value* of 0.65, which showed the best performance in the former experiment. Table 4.3 shows the results. As the results indicate, the highest performance was an F1-score of 0.67 when *gap* was 5 min and the *input window size* was 5 min. When *gap* was less than 10 min, the performance changed similarly, which seems to be because each failure has a different time difference from the fault messages. We predicted that a larger *input window size* would result in a higher performance as more information was entered; however, it was similar when *gap* is under 10 min. This is because the time difference between most fault messages and failures is less than 10 min. However,

Table 4.3: Gap and window size test

Win (min) / Gap (min)	5	10	20
1	Acc: 0.93, Rec: 0.45, F1: 0.56	Acc: 0.94, Rec: 0.60, F1: 0.57	Acc: 0.88, Rec: 0.43, F1: 0.22
3	Acc: 0.93, Rec: 0.37, F1: 0.46	Acc: 0.95, Rec: 0.33, F1: 0.43	Acc: 0.94, Rec: 0.57, F1: 0.44
5	**Acc: 0.95, Rec: 1.00, F1: 0.67**	Acc: 0.95, Rec: 0.33, F1: 0.36	Acc: 0.82, Rec: 0.71, F1: 0.26
10	Acc: 0.93, Rec: 1.00, F1: 0.17	Acc: 0.91, Rec: 0.43, F1: 0.35	Acc: 0.92, Rec: 0.40, F1: 0.24

when the *gap* is over 10 min, the performance is very low.

We analyzed cases of false negatives, which are a failures that the model did not predict. We analyzed two possible reasons for false negative: 1) The fault message did not enter the window. In particular, two failures have a gap of 1 min with a fault message, and for those failures, no model with a gap of more than 1 min did predict. 2) The error log was repeated so frequently that the CNN did not learn the message properly. In this case,the CNN output did not exceed the threshold. The fault message was related to ARB, and the fault message was repeated for a longer period of time than the *input window size*. Learning about overly repeated messages is unlikely to be predicted with a CNN, and a new algorithm to detect such messages is needed.

Finally, for a performance evaluation of the CNN, we compared the performance of a GRU and RNN. Each model is simple, containing 256 cells and fully connected layer with a Sigmoid function. All three models used data with *input window size* 5, *gap* 5, *pre-failure size* 5, and *pre-failure value* 0.65. As a result, the GRU showed an accuracy of 0.53, a recall of 0.95, and an F1-score of 0.41, and the RNN showed an accuracy of 0.53, a recall of 0.68, and an F1-score of 0.33. Fig. 4.3 illustrates the receiver operating characteristic (ROC) curve for an accurate comparison of the three

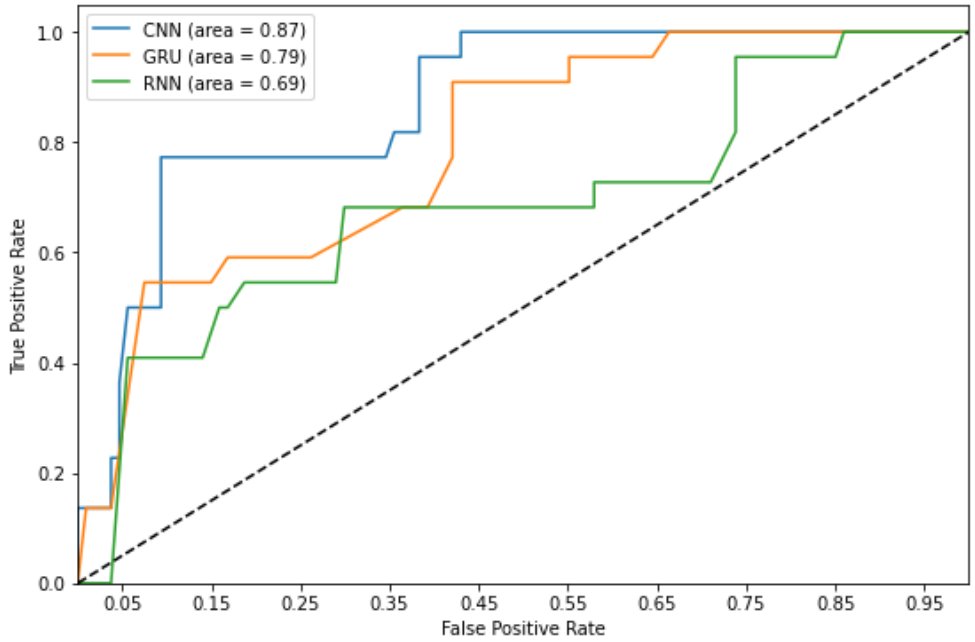

Figure 4.3: ROC curve with CNN, RNN, and GRU

models. The plot shows angular lines because the number of data points is not large. The much higher area under the curve (AUC) of the CNN compared to the AUCs of the RNN and GRU shows that the CNN is much better learned.

V. Conclusion

5.1 Summary

In this study, we proposed a model that analyzes logs extracted from VMs that execute VNFs and determine whether a failure will occur in the future. The proposed model was built by adapting CNN-based sentence classification techniques. To increase the performance of the model, we made word embeddings with a log corpus, and we used the pre-failure tagging method. We generated VM log data in the OpenStack testbed, and confirmed experimentally that the proposed method boosts the performance of the model. Our proposed model was shown to predict failures 5 min in advance with an F1-score of 0.67.

5.2 Limitations and Future Work

As a limitation of this research, we used stress tools to generate failures instead of using real log data. Therefore, we are continuously collecting real log data and planning to use a large amount of data in future research.

We simply removed all numbers from the log; however, some of the numbers are concerned with the error status (e.g., ”ovs timeval(handler38)—WARN—faults: 19 minor, 0 major”). So, learning model should also consider the relationship of number with the words right before or next to it, and reflecting the results in future studies if they are useful for failure prediction.

In addition, in this study, data with large time differences between the fault and failure were excluded from the learning. To learn data with a large time difference, the window size needs to be increased; however, the performance of the CNN decreased and the learning time becomes much longer when the window size exceeds 20 min.

To learn this type of failure, we are currently working on a new learning approach that use CNN results in RNN-based models to understand the sequence of log messages, and it is expected to show higher performance.

Recently, container technology has been supported by a number of companies, such as Google and Alibaba. However, failure detection and prediction studies on container environments are significantly lacking compared to those in VM environments. Since our study has proceeded as log-based, it is expected to be easy to apply to container environments, and the study of failure detection in container environments is also a good future research topic.

요 약 문

본 연구에서는 NFV 환경에서 가상 머신 (VM)에 발생하는 고장을 예측하는 모델을 제안하고 실험을 통해 검증했다. 제안한 모델은 VNF가 설치된 가상 머신에서 추출한 로그를 분석하여 일정 시간이 경과한 뒤에 고장이 일어날지 아닐지를 판단한다. 우리는 기존의 sentence classification 문제에서 사용되는 합성곱 신경망 기술을 적용하였으며, 가상 머신의 로그 분석에 적합한 데이터 전처리, log word embedding, pre-failure tagging 등의 기법을 함께 사용하여 더 높은 탐지 성능을 갖도록 하였다. 우리는 OpenStack에서 생성한 데이터로 제안한 모델을 검증했으며, 그 결과, 제안하는 방법이 5분 이후의 고장을 F1-score 0.67로 예측할 수 있는 것을 확인하였다. 하지만 실제로 발생시킬 수 있는 고장 데이터가 많지 않기 때문에 향후 연구로서 더 많은 데이터를 생성하여 연구에 활용할 예정이다.

References

[1] Network functions virtualization (nfv) white paper. `https://portal.etsi.org/portal/server.pt/community/NFV/367`, 2012. Accessed: 2021-06-12.

[2] Omar Sefraoui, Mohammed Aissaoui, and Mohsine Eleuldj. Openstack: toward an open-source solution for cloud computing. *International Journal of Computer Applications*, 55(3):38–42, 2012.

[3] Jibum Hong, Suhyun Park, Jae-Hyoung Yoo, and James Won-Ki Hong. Machine learning based sla-aware vnf anomaly detection for virtual network management. In *2020 16th International Conference on Network and Service Management (CNSM)*, pages 1–7. IEEE, 2020.

[4] Sukhyun Nam, Jiyoon Lim, Jae-Hyoung Yoo, and James Won-Ki Hong. Network anomaly detection based on in-band network telemetry with rnn. In *2020 IEEE International Conference on Consumer Electronics-Asia (ICCE-Asia)*, pages 1–4. IEEE, 2020.

[5] Qingwei Lin, Ken Hsieh, Yingnong Dang, Hongyu Zhang, Kaixin Sui, Yong Xu, Jian-Guang Lou, Chenggang Li, Youjiang Wu, Randolph Yao, et al. Predicting node failure in cloud service systems. In *Proceedings of the 2018 26th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering*, pages 480–490, 2018.

[6] Haibo Mi, Huaimin Wang, Yangfan Zhou, Michael Rung-Tsong Lyu, and Hua Cai. Toward fine-grained, unsupervised, scalable performance diagnosis for production cloud computing systems. *IEEE Transactions on Parallel and Distributed Systems*, 24(6):1245–1255, 2013.

[7] Seyeon Jeong, Heegon Kim, Jae-Hyoung Yoo, and James Won-Ki Hong. Machine learning based link state aware service function chaining. In *2019 20th Asia-Pacific Network Operations and Management Symposium (APNOMS)*, pages 1–4, 2019.

[8] Suhyun Park, Hee-Gon Kim, Jibum Hong, Stanislav Lange, Jae-Hyoung Yoo, and James Won-Ki Hong. Machine learning-based optimal vnf deployment. In *2020 21st Asia-Pacific Network Operations and Management Symposium (APNOMS)*, pages 67–72, 2020.

[9] Michael Grottke and Kishor S Trivedi. A classification of software faults. *Journal of Reliability Engineering Association of Japan*, 27(7):425–438, 2005.

[10] Guanping Xiao, Zheng Zheng, Beibei Yin, Kishor S Trivedi, Xiaoting Du, and Kaiyuan Cai. Experience report: Fault triggers in linux operating system: From evolution perspective. In *2017 IEEE 28Th international symposium on software reliability engineering (ISSRE)*, pages 101–111. IEEE, 2017.

[11] Ayush Goel, Sukrit Kalra, and Mohan Dhawan. Gretel: Lightweight fault localization for openstack. In *Proceedings of the 12th International on Conference on emerging Networking EXperiments and Technologies*, pages 413–426, 2016.

[12] Qiang Fu, Jian-Guang Lou, Yi Wang, and Jiang Li. Execution anomaly detection in distributed systems through unstructured log analysis. In *2009 ninth IEEE international conference on data mining*, pages 149–158. IEEE, 2009.

[13] Shilin He, Jieming Zhu, Pinjia He, and Michael R Lyu. Experience report: System log analysis for anomaly detection. In *2016 IEEE 27th International Symposium on Software Reliability Engineering (ISSRE)*, pages 207–218. IEEE, 2016.

[14] Weiliang Ji, Shihui Duan, Renai Chen, Song Wang, and Qiang Ling. A cnn-based network failure prediction method with logs. In *2018 Chinese Control And Decision Conference (CCDC)*, pages 4087–4090. IEEE, 2018.

[15] Min Du, Feifei Li, Guineng Zheng, and Vivek Srikumar. Deeplog: Anomaly detection and diagnosis from system logs through deep learning. In *Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security*, pages 1285–1298, 2017.

[16] Yinglung Liang, Yanyong Zhang, Hui Xiong, and Ramendra Sahoo. Failure prediction in ibm bluegene/l event logs. In *Seventh IEEE International Conference on Data Mining (ICDM 2007)*, pages 583–588. IEEE, 2007.

[17] Yoon Kim. Convolutional neural networks for sentence classification. In *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 1746–1751, Doha, Qatar, October 2014. Association for Computational Linguistics.

[18] Ye Zhang and Byron Wallace. A sensitivity analysis of (and practitioners' guide to) convolutional neural networks for sentence classification. *arXiv preprint arXiv:1510.03820*, 2015.

[19] Domenico Cotroneo, Michael Grottke, Roberto Natella, Roberto Pietrantuono, and Kishor S Trivedi. Fault triggers in open-source software: An experience report. In *2013 IEEE 24th International symposium on software reliability engineering (ISSRE)*, pages 178–187. IEEE, 2013.

[20] Tomas Mikolov, Ilya Sutskever, Kai Chen, Greg Corrado, and Jeffrey Dean. Distributed representations of words and phrases and their compositionality. *arXiv preprint arXiv:1310.4546*, 2013.

[21] Google. word2vec. `https://code.google.com/archive/p/word2vec/`, 2013. Accessed: 2021-06-12.

[22] Ye Zhang and Byron Wallace. A sensitivity analysis of (and practitioners' guide to) convolutional neural networks for sentence classification. *arXiv preprint arXiv:1510.03820*, 2015.

[23] Adiscon GmbH. The rocket-fast syslog server. https://www.rsyslog.com/. Accessed: 2021-06-12.

[24] Stress-ng. https://wiki.ubuntu.com/Kernel/Reference/stress-ng. Accessed: 2021-06-12.

[25] The Apache Software Foundation. Apache bench. https://httpd.apache.org/docs/2.4/en/programs/ab.html. Accessed: 2021-06-12.

[26] hping3 (2006). http://www.hping.org/. Accessed: 2021-06-12.

[27] Sukhyun Nam. Anomalypredictionwithlog source code. https://github.com/obiwan96/AnomalyPredictionWithLog. Accessed: 2021-06-12.

[28] TianZhang He, Adel N Toosi, and Rajkumar Buyya. Performance evaluation of live virtual machine migration in sdn-enabled cloud data centers. *Journal of Parallel and Distributed Computing*, 131:55–68, 2019.

Acknowledgements

믿고 응원해주시며 때로는 꾸짖어주신 모든 분들께 감사드립니다

이 논문은 2021년도 정부(산업통상자원부)의 재원으로 산업기술평가관리원의 지원을 받아 수행된 연구임 (No.2009633,초저지연 네트워크 서비스를 위한 SDN기반 인공지능 관제 시스템 개발)

Curriculum Vitae

Name : Sukhyun Nam

Research Interest

Software-Defined Networking (SDN); Network Management; Machine Learning (ML)

Education

2014 – 2019	School of Computing, Korea Advanced Institute of Science and Technology (B.S.)
2019 – 2021	Department of Computer Science and Engineering, Pohang University of Science and Technology (M.S.)

Research/Project Experience

2019. 7. – 2020. 12. 멀티 서비스를 지원하는 프로그래머블 스위치 제어 기술 개발 (Funded by IITP)

2020. 4. – 2021. 8. 초저지연 네트워크 서비스를 위한 SDN 기반 인공지능 관제 시스템 개발 (Funded by KEIT)

2021. 1. – 2021. 8. 인공지능 기반 가상 네트워크 관리 기술 개발 (Funded by IITP)

Publications: International Conference

1. Jiyoon Lim, **Sukhuyn Nam**, Jae-Hyoung Yoo, James Won-Ki Hong, ”Best next hop Load Balancing Algorithm with Inband network telemetry”, 16th International Conference on Network and Service Management (CNSM 2020), Virtual Conference, Nov. 2-6, 2020.
2. **Sukhuyn Nam**, Jiyoon Lim, Jae-Hyoung Yoo, James Won-Ki Hong, ”Network Anomaly Detection Based on In-band Network Telemetry with RNN”, The Fifth International Conference On Consumer Electronics (ICCE-Asia 2020), Seoul, Korea, Nov. 1-3, 2020.
3. Jiyoon Lim, **Sukhuyn Nam**, Jae-Hyoung Yoo, James Won-Ki Hong, ”Load Balancing Algorithm with Programmable Switch”, The 21st Asia-Pacific Network Operations and Management Symposium (APNOMS 2020), Daegu, Korea, Sep. 23-25, 2020

Publications: Domestic Journals

1. 임지윤, **남석현**, 유재형, 홍원기, ”INT 기반 네트워크 이상 상태 탐지 기술 연구”, KNOM Review, Vol. 22, No. 3, December 2019.

Publications: Domestic Conference

1. **남석현**, 홍지범, 유재형, 홍원기, ”로그 및 자원 분석을 통한 VNF 고장 예측에 관한 연구”, KNOM Conference 2021, On-line KNOM Conference Venue, Korea, April 30, 2021, pp. 17-20.
2. 홍지범, **남석현**, 유재형, 홍원기, ”가상 네트워크 관리를 위한 기계학습 기반 이상 탐지 시스템 설계”, KNOM Conference 2021, On-line KNOM Conference Venue, Korea, April 30, 2021, pp. 120-122.
3. 임지윤, **남석현**, 유재형, 홍원기, ”강화학습 기반 링크가중치 조정 로드밸런싱 알고리즘 연구”, KNOM Conference 2020, On-line KNOM Conference Venue, Korea, May 15, 2020, pp. 28-31.
4. **남석현**, 임지윤, 유재형, 홍원기, ”네트워크 텔레메트리 기반 통합 네트워크 관리 시스템 연구”, KNOM Conference 2020, On-line KNOM Conference Venue, Korea, May 15, 2020, pp. 130-132.
5. **남석현**, 현종환, 유재형, 홍원기, ”네트워크 텔레메트리를 활용한 머신 러닝 기반 네트워크 이상 탐지 기법 연구”, KNOM Conference 2019, Daegu, Korea, May. 30, 2019, pp. 75-77.