VM Failure Prediction with Log Analysis using BERT-CNN Model

Sukhyun Nam, Jae-Hyoung Yoo, and James Won-Ki Hong

Department of Computer Science and Engineering, POSTECH, Pohang, Korea
{obiwan96, jhyoo78, jwkhong}@postech.ac.kr

***Abstract*—In this study, we present a failure prediction study of VMs and VNFs in an NFV environment. For the proof of concept, we designed a machine learning model to predict the failure with log analysis and observed the cases where the failure-related logs do not exist in the failed VM, but in the server, or in other VMs operating on the same server. Therefore, in this paper, we propose a model which analyzes the logs of all the related VMs and the server and predicts the possibility that any of the VMs operating on the server will fail. To reduce the huge size of the logs collected from the server and VMs, we propose a pre-processing and tagging method that can improve the performance of our model. In addition, we designed a machine learning model using CNN with BERT, which has performed SOTA in various fields of NLP, to receive logs as input and calculate failure probabilities for the next 30 minutes. To validate the proposed model, we collected failure-related logs and normal logs from an OpenStack testbed, and the experimental result shows that the proposed model can predict the failure of VMs operating in the server with an F1 score of 0.74.**

***Index Terms*—VNF, failure prediction, machine learning, log analysis**

I. Introduction

In the 5G era, network operators are required to provide various advanced services according to rapidly changing stringent requirements. Fortunately, technologies such as software-defined networking (SDN) and network function virtualization (NFV) have been adopted to 5G, allowing service providers to respond quickly to the changes in demand for services and reduce CAPEX/OPEX. In NFV environments, each network function is provided in the form of virtualized network functions (VNFs) running on virtual machines (VMs), so VNF and VM management are getting more important. In particular, when an abnormal state or failure occurs in a VM, the VNF operating in the corresponding VM may not be operated, and service may be disconnected, and thus, it is essential to develop a technology that predicts and responds to this in advance. However, it is not easy to predict and respond to VM failures because there are so many causes and symptoms.

Failure is caused mainly by unexpected errors. Because it is impossible to build on a complete understanding of the entire system when designing a service, several factors (e.g., memory leaks, hardware issues, software bugs, etc.) are often the causes of unexpected errors. In addition, these small errors lead to fatal failures in the process of complex error propagation, making it difficult to pinpoint the cause and reproduce them. Conversely, however, there is an irony that there is a possibility that the error propagation process exists after the error so that we can take action through related symptoms before it leads to failure.

Traditionally, failure management of network devices has been based on log messages that are generated unstructured in real-time in network equipment. Logs are one of the data that best represents the state of equipment, but most systems output logs that are unsystematic and difficult to analyze automatically, so experienced managers are required. For this reason, some researches are being conducted to automatically analyze logs of computing equipment and predict and detect failures or abnormal conditions based on them [1]–[4], but it remains a very difficult problem.

With the development of machine learning, various problems that were difficult to solve in the past have been solved. We can apply machine learning, particularly the techniques used in Natural Language Processing (NLP) in log analysis. In our previous work [5], we used Word2Vec word embedding [6] technique, which is a part of NLP, to convert each word in logs to the numerical vector to use them as input for the machine learning algorithm. In this study, we utilize Bidirectional Encoder Represents from Transformers (BERT) [7], a language model that has recently shown better performance than word embedding in the NLP area.

In our previous work, we found many cases where the failure-related logs of VM did not exist in the failed VM but in other VMs operating on the same server or in the server logs. This is because the server is a complex system, and the error or overload of some VMs or the server may lead to bugs or overload in other VMs. So, in a few cases, a VM failure could not be predicted only by the corresponding VM log. In this study, we propose a model that predicts the probability that any VM operating on each server will fail. Our model predicts server failure $2 \sim 30$ minutes in advance using all logs of the VMs operating on the server and the server logs. The occurrence of server failure means that there is a high probability that a VM on that server will fail. In this way, we could confirm that the prediction performance was higher when targeting the server rather than an individual VM.

This paper includes the following improvements compared to our previous study [5] that used word embedding to predict individual VM failures operating in NFV environments.

1) We use BERT instead of word embedding to reflect the context of the log and respond to a new type of log.

978-3-903176-51-5 © 2022 IFIP

2) We propose a machine learning model to predict server failure instead of individual VM failure since the VM failures are not always caused by the corresponding VM.

Through these two improvements, our new model could predict the failure of VMs in servers with higher accuracy.

II. Related Work

A. Word Embedding and BERT

In most NLP tasks, many studies use words in a converted state, which is called word embedding and is generally used as input for NLP models. So, it is important to keep the meaning of each word in conversion. Previously, pre-trained word embeddings based on large corpus such as Word2Vec [6] were mostly used. However, the word embedding methodology causes biased results in the pre-training corpus because it does not change the embedding after pre-training. To improve this, a language model has been proposed to grasp the context in the training stage even after pre-training and output different embeddings even with the same word. Among them, BERT [7] is a pre-trained language model released by Google in 2018. BERT is evaluated as the most influential language model by recording state-of-the-art 11 NLP tasks. BERT can achieve high performance in completely different tasks by conducting fine-tuning, an additional training process for parameter readjustment, according to the task to be solved.

BERT divides words into smaller sub-words, uses them as tokens, and creates embeddings for each token. BERT uses position embedding and special token called [CLS] to improve performance on different tasks. Google makes public two pre-learned BERT models, BERT-Base and BERT-Large, and in this study, we fine-tuned the BERT-Base model with our log data and used it. The BERT-Base model includes 12 layers of Transformer encoders, each with a hidden layer of 768 dimensions.

B. Failure Prediction and Anomaly Detection

Ji *et al.* [2] predicted the occurrence of network error messages based on preceding log messages in simulation environments. They utilized Word2Vec to convert logs to numeric vectors and used embedding vectors as inputs of the Convolutional Neural Network (CNN) to predict whether errors or failures would occur after a certain period of time. They compared the performance by conducting three machine learning models: Gated Recurrent Unit (GRU), Long Short-Term Memory (LSTM), and CNN, and CNN showed the highest performance. Due to the limitations of the aforementioned word embedding-based research, this study did not show high performance (accuracy was 0.75 when gap lines between the prediction object and the input log window are 2000).

In addition, there is also a study that detects abnormal conditions based on the occurrence of log keys which are defined with log corpus by finding specific patterns without converting logs into vectors [4]. This study used Deep Neural Network (DNN) to learn log patterns in normal situations, and their model detected anomalies with the performance of F1-score 0.98 for log datasets generated with OpenStack. However, these techniques are not preferred over word embeddings or language model-based methodologies because of the problem of scalability that requires analyzing new patterns and finding log keys as the system being used changes.

Our previous study [5] proposed the CNN-based model that predicts VM failures based on log messages and we tested them in an OpenStack testbed. We used Word2Vec to use logs as input. We installed a separate state-checker VM to record the failure state of each VM in the data collection step and sent periodic pings to each VM to check that the current network function is not paralyzed. At the same time, we collected logs from all VMs and make a word embedding model for log analysis with Word2Vec. We trained a CNN model to predict whether the VM would fail after a certain period (gap) of time by receiving the word embedding matrix from collected logs as input and whether each VM fails or not as output.

To improve the performance of the model, we checked whether there was a log related to the failure before each moment was recorded as a failure, and if there was no involved log, we excluded that failure record from VM failure history. In addition, since failure-related logs may exist earlier than gap + input window size before the failure moment, we proposed pre-failure tagging, which tags some windows before failure as semi-failure and label output values as between 0 and 1, so that prevent CNN from learning them as normal state-related logs. Our model shows an F1 score of 0.67 for failure prediction before 5 minutes. Also, we compared our CNN-based model with Recurrent Neural Net (RNN) and GRU, and CNN showed the highest performance. In our new study, made changes to the data labeling method to learn all the failure-related logs, which will be explained in the next section.

III. Data Collection

In this study, we used both the data collected in our previous study [5] and the data collected from the new testbed. Since the two testbeds are similar in several aspects, we will explain the commonalities and differences and explain the two datasets.

A. Testbed

Both the old testbed and the new testbed were configured based on OpenStack. Figure 1 represents testbeds. The controller node manages the network connection between each VM of the compute node and establishes SFCs. The compute node operates servers which include VMs. The old testbed originally included 3 servers in the compute node, but only the logs of two servers in the old testbed were used in this study because one server failed during the data collection process, and the logs were also lost. Each VM runs a single VNF or web server. In addition, we set up the state checker VM to check VM failure every minute with the ping protocol. The monitoring node serves to store the logs and failure history data. Logs were collected from each VM through rsyslogd [8], and they include kernel logs, application logs, systemd logs, and server application logs like Neutron logs and OVS logs. The VM failure record was stored in the form of a yaml file to record the time when the failure occurred.

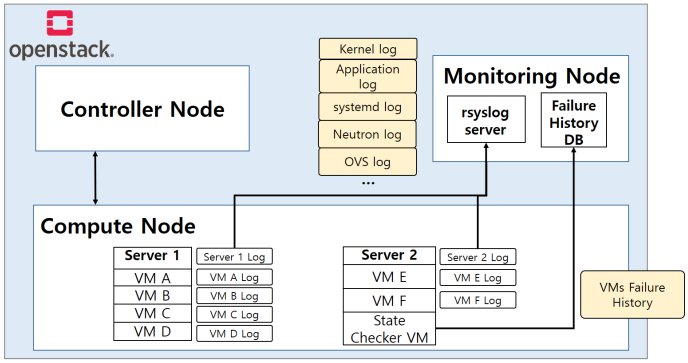

Fig. 1. Data collection testbed structure

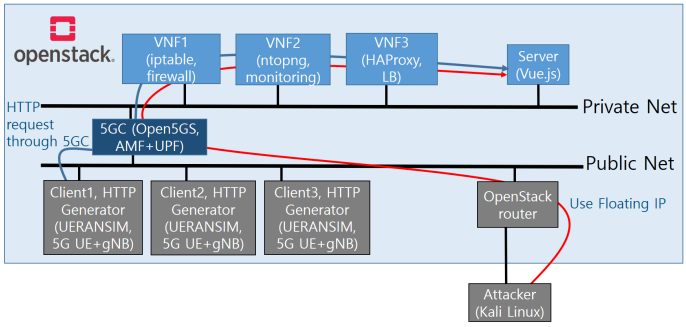

Fig. 2. One of traffic generation scenarios

Figure 2 represents a traffic generation scenario for data collection in a new testbed. We constructed a 5G network environment in a new testbed for the diversification of output logs and realistic scenarios. We constructed the 5G network using Open5Gs [9], an open project to configure 5G private networks, and all virtual clients in our scenario had to set up 5G UE and RAN via UERANSIM [10] to send HTTP traffic to the web server. In the old testbed, we allowed HTTP traffic to be sent directly to the web server through an SFC without the connection to 5G core systems. For the HTTP traffic generation, we used Apache Bench [11] in the old testbed, and Web-traffic-generator [12] in the new testbed. Through the Web-traffic-generator software, we could make a scenario to access the web page and then click randomly selected buttons to move around the web homepage. To this end, we used Vue.js to implement a simple Internet shopping mall in a server VM. When the Web-traffic-generator software reaches the maximum depth, it waits for a certain period (within 30 seconds) which is set randomly and sends new HTTP traffic.

In both old and new testbeds, we allowed HTTP traffic to go through multiple VNFs configured with SFCs before reaching the server. We flexibly organized SFC with several VNFs, among firewall (iptable), intrusion detection system (Suricata), monitoring application (ntopng) deep packet inspection (nDPI), and load balancer (HAProxy), and Figure 2 shows the one example of set SFC.

In addition, if only normal traffic is generated, VMs are rarely failed, so we inject malicious attacks to overload and fail the VMs. In the old testbed, we used a simple DDoS attack using the DDoS tool hping3 [13], but in the new scenario, we generated the latest DDoS attacks like Slowloris, RUDY, Hulk DoS using the multi-purpose hacking tool Kali Linux [14]. These attacks overloaded the VMs operating the web server and VNFs and caused failures. In addition, we overloaded CPU and memory usage for each VM with Stress-ng, a resource overload tool [15].

B. Pre-processing

We used not only the log of a single VM, but also the logs from VMs and the server as input, so the number of input logs became too large. In particular, OpenStack-related applications (eg. Neutron, Nova, OVS) which operated on the server generated almost 10 lines of logs per second. Aside from the performance of the model, it took a long time just to read the logs, so the pre-processing to reduce the number of logs is inevitable.

Some logs are needed to remove unnecessary information based on rules. For example, the kernel log periodically outputs the addresses of major registers, so we removed this information by rule-base. This analysis can be easily performed even if there is a new type of log because it is a simple process to find unnecessary patterns in the log without background knowledge.

To reduce the number of logs further, we decided to find words that usually appear a lot before errors occur, and based on them, we would leave only logs containing those words. To this end, we collected logs within 30 minutes before the failure occurred, listed the output frequency of each word in high order, and we removed general words that did not seem to be related to the errors, leaving words related to the error. For example, words such as 'user' and 'system' were excluded because they also appeared in general logs, and 'timeout' and 'out' appeared frequently, but we left only 'timeout' as an error-related word. Also, we left abbreviations such as oom (out-of-memory) and err (error). As a result, the following 13 words remained error-related words. And we leave the logs only if they include error-related words. Since these words are common words that occur regardless of the system, they can be applied equally to the log of the new system.

- reset, fail, not, failure, timeout, kill, err, oom, stop, exit, restart, long, warn

After the number of logs decreased with error-related words, we applied some additional pre-processing processes. We extracted time and application information from the log and removed all numbers and symbols and changed capital letters to lowercase letters. In addition, we used the extracted time information by sorting all logs in the input window in chronological order. Furthermore, we had to prevent the same log from being present in the input window during the experiment. Therefore, when using the input window as an input to the model, we checked the first five words for each sentence, and if there is another sentence that has the same first five words, we removed it from the input window.

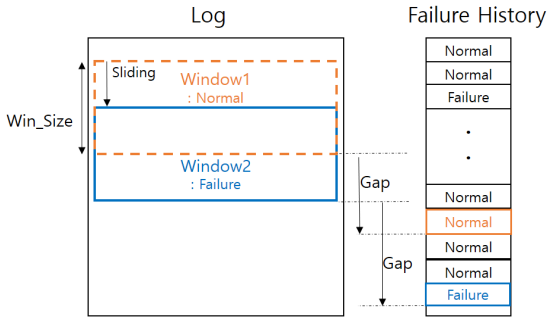

Fig. 3. Basic concept of labeling method

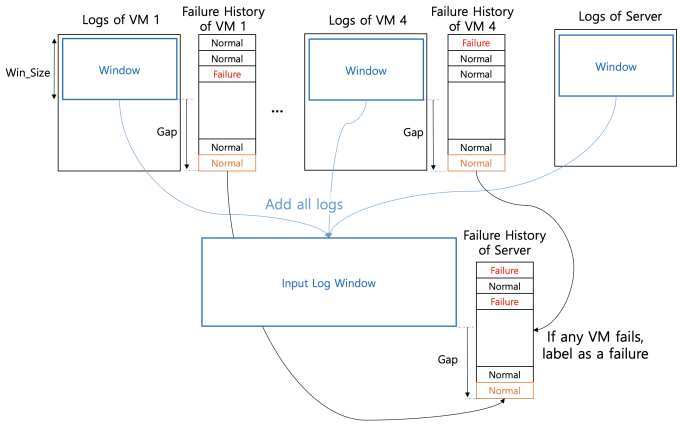

Fig. 4. Labeling method for server failure prediction

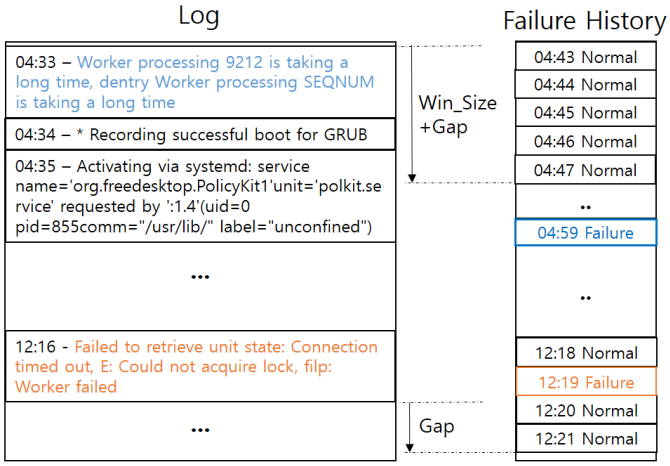

Fig. 5. The problem of labeling method in [5]

C. Data labeling

For supervised learning, inputs and corresponding outputs are required. In our model, we use sliding input windows to get input sequences and label each window to denote whether related to a failure or a normal state. Figure 3 shows the basic concept of the labeling method. Both log and failure record data collected from the testbed have time records. A window with a size of 'input window size' slides the log data and generates input windows. In this case, the amount of input sequence may be adjusted by adjusting the sliding window size. In this study, we used 10 minutes as input window size and 5 minutes as sliding size. Logs within the input window become input data. Each input log is labeled as a failure if the failure history shows failure after the 'gap' period from the end time of the input window and is labeled as normal if it is normal after the gap.

In the meantime, as mentioned before, there were cases where the failure-related logs occurred on the server log or other VMs, not the failed VM. In particular, 14 of the 53 failures collected in our testbed have related logs in the server or other VMs, not failed VM. In these cases, the input window which is labeled as a failure does not contain the log sequence related to the failure, and the input window which includes the log related to the failure is labeled as normal, so the model is not learned properly, so we targeted to predict the server failure. We used logs from multiple VMs and the server together, and if any of the VMs operating on the server fail, we labeled the input log window as a failure (Figure 4).

Also, in this study, we changed the input window size only for the windows that are labeled as a failure. In our previous study, there was a problem that as the gap and window size were changed, only some of the total failure data was used. For example, in Figure 5, if we train a model that predicts a failure after 5 minutes by receiving a 10 minutes log window as input, all windows containing 4:33 are labeled as normal because 4:38 through 4:47 is normal. The failure occurred at 4:59 which is 26 minutes after the failure-related log. Conversely, failure-related logs could occur just before a failure. Gap (5 min) after the failure-related log in 12:16, is 12:21, and the failure history of that time is normal, so the model can not learn properly for failure occurred at 12:19. Among the actual collected data, there were 11 cases where the gap between the failure and the failure-related log was more than 20 minutes, and 13 cases where the gap was less than 3 minutes.

In order to label both of these cases normally, there is no choice but to significantly increase the input window size and reduce the gap. But if we make the input window size large for all windows, then the size of the input sequence becomes large and learning becomes difficult. So we used some tricks here. We set the input window size of the normal window as 10 minutes as before, and change only the failure window to have an input window size of 28 minutes. So that all logs from 30 minutes to 2 minutes before the failure occurred time were used as input. We determined that a minimum time interval with the failure prediction point was necessary to take action after the failure prediction, and decided not to include cases where the gap was less than 2 minutes. In most cases, the gap was less than 30 minutes, so we set the maximum value as 30 minutes. Accordingly, our model was trained to calculate the probability that the failure occurs within 30 minutes, after 2 minutes from the last log. With the changed method, both cases in Figure 5 are included in the failure-indicating input log window.

However, since we set the input window size to 10 minutes,

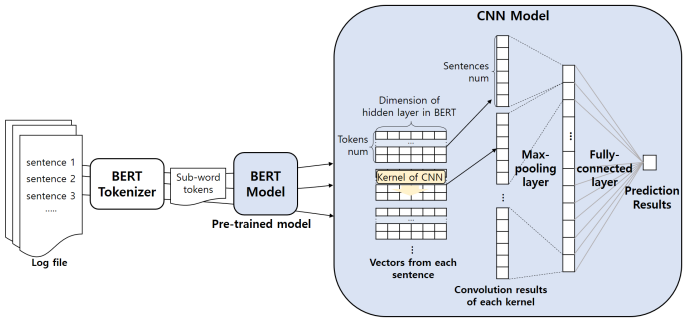

Fig. 6. Proposed BERT-CNN model

there is a problem that the window size of the normal window and the failure window is different, and if the window size is different, the amount of input log is different. But CNN learns to extract only important data from large amounts of data. Since we used the max-pooling layer as CNN's pooling layer, if the CNN model is learned normally, it can only find messages related to failures regardless of the amount of input data. So the size change of the input data is irrelevant, and this labeling method could improve the performance of the model and we verified it.

IV. Model Design and Implementation

Our proposed BERT-CNN based server failure prediction model gets logs from VMs operating VNFs and servers in an NFV environment and utilizes them for failure prediction. For failure prediction, we use the CNN model to learn whether or not the failure exists within a certain period by receiving the output embedding of the BERT model as input.

The detailed structure of the entire model is shown in Figure 6. Only the colored elements in the figure are factors that change parameters through learning. First, the BERT tokenizer tokenizes the words in each sentence into sub-words. We use the pre-trained and opened to public BERT model named BERT-Base model, which contains 768 hidden layers, thus outputting 768-dimensional vectors for each token. BERT could be fine-tuned with backpropagation in the training process, and the overall hyperparameters of BERT are modified to be more suitable for log analysis. BERT creates a matrix of the 768 $\times$ number of tokens within the log, which is used as input to CNN.

The output embedding of the BERT model is received as input to the CNN model to predict whether a failure occurs within a certain period (2-30 min). In the convolutional layer, the kernel moves the embedding matrix and performs the convolutional operation. We used the kernels with certain kernel sizes, and each kernel strides by the vector size (768) so that the kernel does not move within a single token. We used kernel sizes 3, 4, 5, and 6 and 50 kernels of the same size were used each. In the convolutional layer, we used a Rectified Linear unit (ReLU) as an activation function. As a result of the convolutional layer, as many vectors as the number of kernels are generated, and the max-pooling layer leaves only a maximum value through each vector. After going through the pooling layer, a vector with the size of the number of kernels (we used 200 kernels, so 200-dim) is generated.

The result of the pooling layer goes through the drop-out layer with 0.2 of drop-out rate and is put into a fully-connected layer. The fully connected layer uses Sigmoid as an activation function to output a single value. The generated value is a value between 0 and 1, and the closer to 1, the higher the probability of failure, and if the value is over the threshold of 0.5, then the decision of the model will be a failure. To learn this, log data tagged with 1 in the failure state and 0 in the normal state are used in the learning process. We used binary cross-entropy loss (BCE loss) as a loss function, which uses to learn how close we are to the prediction target in classification problems. We used Pytorch's BCEWithLogitLoss function, which combines Sigmoid and BCEloss, for computational advantage.

In the learning process, we confirmed that the model was very sensitive to the learning rate. Therefore, we did not use a static learning rate in the learning process, but changed the learning rate through the learning rate scheduler and used it for learning. We used Pytorch's *get_cosine_schedule_with_warmup* scheduler, which linearly increases the learning rate from 0 to the initial learning rate of the optimizer during the warmup step, and decreases with the cosine function. Through this, when the loss value reaches a certain minimum, it is not allowed to deviate again.

In the experimental stage, we experimented with the Word2Vec-based model proposed in the previous study to compare the performance of our proposed model.

V. Experiments and Evaluation

We observed 39 failure situations in the old testbed for 2 months. In the new testbed, we observed 11 failure situations for 2 months. We could not find why the new testbed had fewer failures, even though we generated overload to two testbeds similarly, but we analyze it as the cause of the failure situation is various and it is difficult to reproduce the failure. Of these, 8 failure situations were not used in actual experiments because there was no log associated with the failure, or the failure-related logs were outside the period we set (30 minutes to 2 minutes before the failure). Because these failures are not subject to our prediction and require analysis of other data (resource usage, etc) than logs, predicting them remains as a future study.

Table I represents the data used in this study. There is a total of four servers subject to failure prediction, with two in the old testbed and two in the new testbed. We will refer to old-1, old-2, new-1, and new-2 servers, respectively. Among them, the new-2 server includes a VM that operates 5GC and a VM that operates a Vue.js-based web server, and these applications have not been run on other servers. In addition, three of the six failures that occurred in the new-2 server occurred in that applications, and a failure-related log was found in those applications logs, so these logs are different from other failure-related logs. Therefore, if we include new-2 server data in test

data, we could determine whether our proposed model can detect failures that have never been learned types. Therefore, we made data from old-1, 2 servers as training data, and data from new-1, 2 servers as test data. And we added normal logs of applications driven by new-1, 2 servers during 2 days in training data so that the model was learned for normal logs from Vue.js and 5GC.

We first conducted an individual VM failure prediction experiment, and then do a server failure prediction experiment, to see if our proposed server and VMs-based VM failure prediction method is better suited for the VM failure prediction task. In addition, for the performance comparison between the word embedding-based model and the BERT-based model, we also trained a model proposed in a previous study [5], that converts each word in the input logs to word embedding using Word2Vec word embedding and use them as input of CNN. We used the same data for the Word2Vec-based model and the BERT-based model. We did not use a public Word2Vec model but used a custom word embedding model which is generated using one month of log data among the data we collected, so is more suitable for log analysis.

A. Individual VM Failure Prediction

As mentioned in section III, when conducting a failure prediction experiment for individual VMs, some failure situations cannot be included in the data because the failure-related logs exist on other VMs or servers. Therefore, 24 failure situations were included in the training data, and 5 failure situations were included in the test data. The normal situations were too many compared to the failure situations, so we undersampled the normal data to 1/10. As a result, 3,817 normal data were included in the training data, and 468 normal data were included in the test data.

Figure 7 shows the experiment results for the individual VM prediction task. We trained our model for 200 epochs. As can be seen from the graph, the loss value of the Word2Vec-based model decreased faster than the BERT-based model at the beginning, but thereafter, the loss value decreased poorly, and eventually showed much lower performance than the BERT-based model. In terms of performance, the model using Word2Vec showed lower performance with an F1 score of 0.59 and the model using BERT showed a performance of an F1 score of 0.65. In our previous work [5], the Word2Vec-based model performed failure prediction with an F1 score of 0.65 on the old data, but the same model performed much lower on our new data because it includes new types of application in test data, but our BERT-based new model performs prediction with still high performance for new data.

TABLE I
DATA USED IN EXPERIMENTS

TB	Server Name	Num of Failures	Used in	Running VNFs
old	old-1	24	Train set	HAProxy, iptables, nDDPI, Snort
old	old-2	10	Train set	Suricata, ntopng
new	new-1	5	2 days of normal data in train set, rest (normal, failure) in test set.	HAProxy, iptables
new	new-2	6	2 days of normal data in train set, rest (normal, failure) in test set.	**5GC, Vue.js (unseen in training data)**, nDPI, ntopng

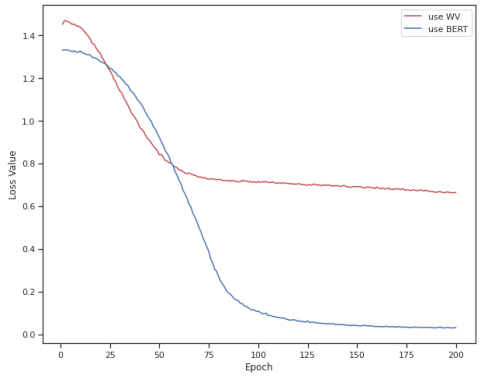

Fig. 7. Experiment results for individual VM failure prediction

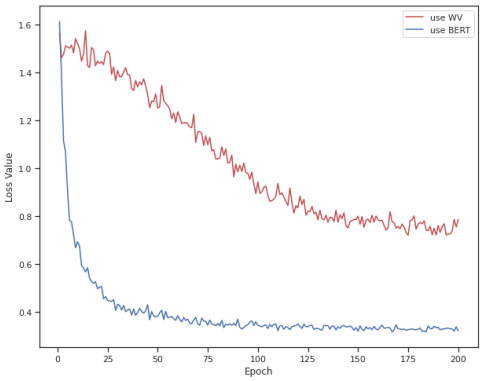

Fig. 8. Experiment results for server failure prediction

B. Server Failure Prediction

We trained our model for 200 epochs for server failure prediction experiment. Figure 8 shows the experiment results. The results show that the loss value fluctuates much more than the failure prediction experiment for individual VM. This is analyzed because the number of data was significantly reduced due to a memory shortage problem. However, the final performance is much higher. The model using Word2Vec showed performance with an F1 score of 0.64 and the model using BERT showed a performance with an F1 score of 0.74, which is much higher than results in individual VM failure prediction. This shows that it is correct to predict VM failures based on all VMs and the server, not only individual VM logs.

We also compared our proposed BERT-CNN model with another simple machine learning model to determine if the CNN is suitable for log analysis. Figure 9 represents the Receiver Operating Characteristic curve (ROC curve) of a model using the BERT model and the GRU model together, a model using Word2Vec and CNN together, and our proposed model. We implemented a BERT-GRU model in the form of GRU receiving the output of BERT as input, and we implemented a simple form of GRU with one hidden layer

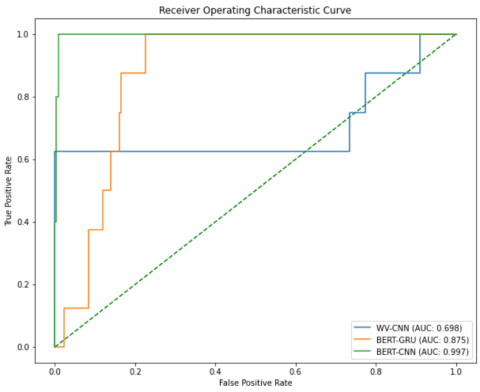

Fig. 9. ROC curves of several models

with a hidden dimension of 20. Like before, we trained the BERT-GRU model for 200 epochs, and draw the ROC curve using the prediction result at the best performance at the test data. Since all three ROC curves were made with only 10 failure data in the test data set, they are somewhat bent. When comparing the Area Under Curve (AUC), it can be seen that our proposed model is the best learned. In particular, the BERT-CNN model has an AUC of nearly 1, which means that further lowering the threshold value can result in a much higher F1 score. We confirmed that setting the threshold to 0.4 can increase to F1 score to 0.86.

There is also a disadvantage of the BERT-based model, which, unlike the Word2Vec-based model, takes a much longer time to train because the BERT model also learns during the training process. On average, the Word2Vec-based model took 48 seconds per epoch, while the BERT-based model took 1 minute and 56 seconds on average, requiring more than double the time in the training step. For this reason, we judge that a Word2Vec-based model, rather than a BERT-based model, would be suitable in cases that require fast training than high performance.

VI. Conclusion and Future Work

In this work, we proposed a machine learning-based model that predicts the failure of VMs operating VNFs in an NFV environment. In the process of collecting fault and log data, we have observed some cases where failure-related logs exist in other VMs or servers, so we proposed a server failure prediction model that calculates the probability of VMs failure by inputting the log data from whole VMs in server and server log data. In the proposed model, we used the BERT model that achieves SOTA in various fields in NLP to improve the performance of log analysis and used CNN to use the output of BERT and predict the failure between 2 and 30 minutes from the input log. Experimental results showed that our model predicts server failures with the performance of an F1 score of 0.74 for test data including failure data that are unobserved in training data and demonstrates the superiority of this model by comparing it with other models. Thus, our model can be easily applied to real-world networks.

As future work, we are planning to improve the prediction performance of our model by utilizing a Generative Adversarial Network (GAN) that learns together the generator that generates fake data to address the shortage of training data that the model can utilize.

Acknowledgment

This work was supported by Korea Evaluation Institute Of Industrial Technology (KEIT) grant funded by the Korea Government(MOTIE) [(No.2009633) Development of AI network traffic controlling system based on SDN for ultra-low latency network service].

This work was supported by the Institute of Information & Communications Technology Planning & Evaluation (IITP) grant funded by the Korean government (MSIT) (2018-0-00749, Development of Virtual Network Management Technology based on Artificial Intelligence).

References

[1] 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.

[2] 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.

[3] 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.

[4] 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.

[5] Sukhyun Nam, Jibum Hong, Jae-Hyoung Yoo, and James Won-Ki Hong. Virtual machine failure prediction using log analysis. In *2021 22nd Asia-Pacific Network Operations and Management Symposium (APNOMS)*, pages 279–284, 2021.

[6] Tomas Mikolov, Kai Chen, Greg Corrado, and Jeffrey Dean. Efficient estimation of word representations in vector space. *arXiv preprint arXiv:1301.3781*, 2013.

[7] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. Bert: Pre-training of deep bidirectional transformers for language understanding. *arXiv preprint arXiv:1810.04805*, 2018.

[8] Adiscon GmbH. The rocket-fast Syslog Server. https://www.rsyslog.com/. [Online; accessed 04-July-2022].

[9] Inc. NextEPC. Open5GS. https://github.com/open5gs/open5gs. [Online; accessed 04-July-2022].

[10] ALİ GÜNGÖR. UERANSIM. https://github.com/aligungr/UERANSIM. [Online; accessed 04-July-2022].

[11] The Apache Software Foundation. Apache Bench. https://httpd.apache.org/docs/2.4/en/programs/ab.html. [Online; accessed 04-July-2022].

[12] web-traffic-generator. https://github.com/ReconInfoSec/web-traffic-generator. [Online; accessed 04-July-2022].

[13] 2006 Salvatore Sanfilippo. hping3. http://www.hping.org/. [Online; accessed 04-July-2022].

[14] OffSec Services Limited. Kali Linux. https://www.kali.org/. [Online; accessed 04-July-2022].

[15] Stress-ng. https://wiki.ubuntu.com/Kernel/Reference/stress-ng. [Online; accessed 04-July-2022].