Virtual Machine Failure Prediction using Log Analysis

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

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

***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 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. The experimental results showed that the proposed model can predict failures before 5 minutes with the F1 score of 0.67. The proposed model will be used for VM proactive live migration to avoid service degradation and interruptions caused by failures.**

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

I. Introduction

Today's networks are becoming larger in size and more complex in structure. The advent of software-defined networking (SDN) and network function virtualization (NFV) technologies has reduced CAPEX/OPEX, but the complex virtual structures have made it more difficult to monitor and take action on virtual network and server failures. To address this problem, many studies are underway for anomaly detection in SDN/NFV environments, but there is a lack of research on technologies that proactively predict failures in servers, virtual machines (VMs), and virtual network functions (VNFs) to take action before failures occur.

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 usually highly unbalanced. For example, Microsoft cloud services reported that each day, less than 0.1% of nodes encounter failures [2]. 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 [3]. However, most failures have associated symptoms, and recent advances in machine learning technologies have made it possible to analyze huge numbers of complicated data.

Mandelbug [4] 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 [1], network faults are more likely to be of 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 Mandelbug is longer than that of regular bugs [1]. 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 [**?**]. 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 of computer equipment can be found in the log. Most server and network equipment provide real-time log output (e.g., syslog). Some studies are underway to harness logs in network management [5]–[8]. However, as the software structure of servers and networks has become more complex, the amount of logs has increased proportionally. Since most log data is not systematically generated, automatic analysis of log data remains a fairly challenging task.

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 [9].

In this paper, we propose a CNN based VM failure prediction model which uses logs extracted from each VM as input values. The model predicts the failure with former fault messages. The model includes pre-processing techniques for 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 a month in an NFV environment. The experimental results showed that the F1 score was 0.67 to predict the failure in 5 minutes.

II. Related Work

A. 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. In general, faults are classified as Bohrbug and Mandelbug faults [4]. 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 [4]: 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 [10], LAG and ARB correspond to the second case, which has a delay in failure occurrence.

- ARB (Aging Related Bug) : A kind of bug that can cause an increasing failure rate and/or degraded performance, known as software aging. This represents a situation in which the error state is generated slowly due to overloads, like continuous memory leaks or an increase in total system runtime.
- LAG : A kind of bug that is non-aging related Mandelbug (NAM), but 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.

B. Sentence Classification and Log Analysis

Y. Kim [9] proposed to use a CNN model in a sentence classification problem. This work used the word embedding [11], a technique for representing words as dense vectors to use sentences as input features for CNN. The author generated sentence vectors via a concatenation process with generated word embedding vectors. The generated sentence vector was input through a convolution layer. The filters used in the convolution layers used the dimension of the product of the filter size h and size of embedding vectors k, that is hk. Because the length of the input sentence is varying, the dimension of the vector generated at this point is not constant. So max-pooling layer takes the maximum value for each vector generated from a filter. 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 labels. Although the proposed CNN model's performance 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.

Du *et al.* [8] 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.

W. Ji *et al.* [7] proposed a CNN model that collected log data from wireless communication systems in a simulation environment to predict whether log data after a constant gap contained failure messages. In this work, the authors used the pre-processing technique which removed both symbols and numbers in the log data. The CNN model was based on the model in [9]. This work was compared to models using long short-term memory (LSTM) and gated recurrent units (GRU) for performance verification of the proposed CNN model, where the result showed that the CNN model showed slightly higher performance than the LSTM and GRU model. The learning results showed an accuracy of about 0.75 when the gap was 2000, and an accuracy of about 0.57 when the gap was 5000. The problem is that 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, since 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 [9], but we added pre-failure tagging steps in the data tagging stage to generate more suitable data for log analysis.

III. Design and Implementation

This section describes the design of the proposed failure prediction model and CNN model.

A. CNN based VM failure prediction model

We propose a model that uses log data to predict and alert failure risks in VM. The proposed model uses a time-based gap until the prediction point. The proposed CNN based VM failure prediction model structure is shown in Figure 1.

As a testbed for collecting log data, we used NFV infrastructure (NFVI) built using OpenStack. This NFVI environment operates a variety of VNFs (e.g., IDS, firewall) to provide network services. Each VM passes logs from VNF, system daemon, and kernel to the monitoring node using rsyslog [12].

The monitoring node extracts data of a pre-defined *input window size* from the collected log data using timestamp (i.e. logs in the last 10 minutes). Extracted data go through the pre-processing step and is transmitted to the AI node. AI node contains pre-trained word embedding and CNN model. In the AI node, the log data passed from the monitoring node is converted to word embedding vectors and used as input for

the CNN model. CNN calculates the probability that a failure occurs after a *gap* of minutes based on input and passes it to the control node when that probability crosses the predefined threshold. The control node can then move the VNF to another server to prevent the failure before the failure occurs through VM live migration.

B. 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. 2 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 [13] 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."

C. Convolutional Neural Network

We build a CNN model based on the CNN model from [9]. Figure 3 represents the CNN model and inputs and outputs used in our work. 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 1. According to [14], 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.

D. Output Tagging

Learning the CNN model requires a label for each input window. Similar to the general classification problem, we tagged each input window to 1 if it was related to failure, and 0 if the state would be normal. We tagged based on failure history, which is data recorded every minute whether each VM was failed or in a normal state. Each input window was tagged whether a failure occurs after the *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. 4 shows the difference between regular tagging and pre-failure tagging. In this 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*.

To ensure that the changed tagging method is applied to learning, we utilize the following KL divergence-based loss function. y_{true} represents the tagged values, and y_{pred}

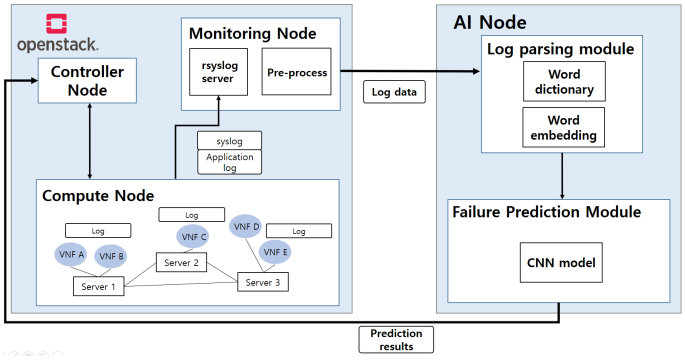

Fig. 1. CNN based VM failure prediction model

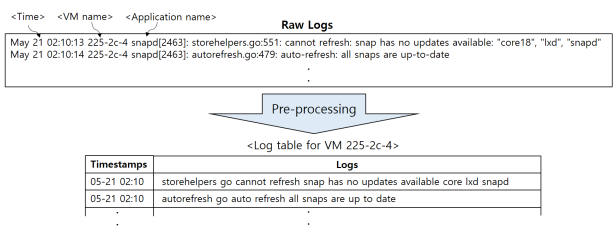

Fig. 2. Pre-processing example

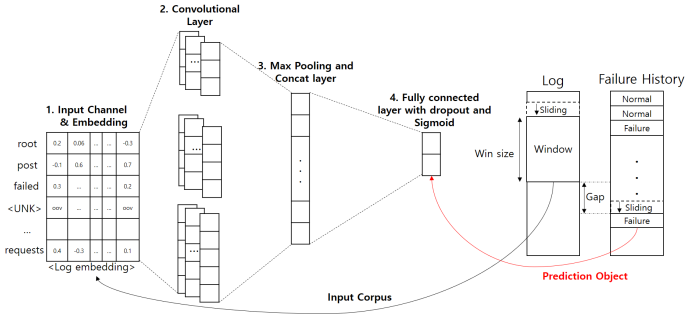

Fig. 3. CNN model design

represents the output of CNN. We also applied class weights to handle unblanched data, which will be explained later.

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

With the pre-failure tagging method, windows near failure will output values that are close to *pre-failure values*, but must be tagged as normal. So we set the threshold of failure as *pre-failure value*+0.05 so that the window's output does not exceed the threshold.

IV. Experiment and Evaluation

In this section, we describe the experiments we conducted to verify the suitability of the proposed model and CNN models and describe the results.

A. Data Collection

We conduct experiments by building the NFVI described in Chapter 3. The NFVI consists of three servers that take on compute nodes and two servers take on for monitoring node and controller node respectively. Each compute node also contains VMs to run VNF and to serve as clients and servers. We installed six different VNFs (Suricata, HAProxy, iptables, ntopng, nDPI and Snort) on each VM to output various logs, and also to create different situation. We changed the specification of each VM in response to each VNF's requirement.

We generated multiple client–VNF-server chains that utilize each VNF as a single service, allowing each VNF to handle the

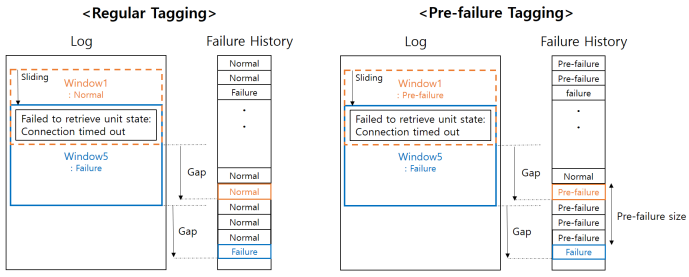

Fig. 4. Regular tagging and pre-failure tagging

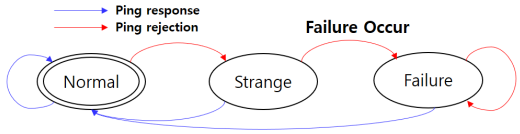

Fig. 5. VM state tagging DFA

traffic. We used the Apache web server as the server. We need to check the status of VMs, so we sent a ping message from the other server to each VM every minute to check the status. We decided to tag the VM as a failure when it refuses to ping for one minute in a row. Figure 5 shows the deterministic finite automata (DFA) representing the tagging scheme. In addition, when multiple VMs became a failure state at the same time, it was excluded from the VM failure tag, judging that there was a problem with the server, not with the VM.

We collected data for two weeks, and since failures usually do not occur easily on VMs, we overload them in three ways to generate as many failures as possible.

- Resource overload – We overloaded CPU and memory for each VM with Stress-ng [15], a resource overload tool. We continuously increase CPU usage and memory usage. The overload started at 50% and increased 5% every 5 minutes, and when a failure occurred, overload figures have been reset.
- Traffic overload - Client VM sent traffic requests to the server with Apache Bench [16], a server performance checking tool. It continuously increases the number of requests and increases the number of concurrent connections every 30 minutes. If a failure occurred, overload figures have been reset.
- External attack – The attacker server generates a DDoS attack to a randomly picked VM. It transmitted more than 100,000 packets at intervals below 6μs. Transmission packet interval, attack time were randomly selected 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 [10].

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.

B. 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 experimented to find 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 catch the lower bound. According to [17], live migration in OpenStack includes 9 steps, and the original VM services until the 6th step, stop-and-copy, which is the step that VM is paused. Therefore, we determined that the *gap* should be longer than the time that it takes to get to the 6th step. We tested how long it would take to perform five steps. As a result, it took an average of 45 seconds before the stop-and-copy phase, based on VMs with the size of 5GB on OpenStack. So we decided that even if the *gap* is one minute, migration could occur before the failure occurred in 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.

TABLE I
PRE-FAILURE SIZE AND 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

TABLE II
GAP AND WINDOW SIZE TEST

Gap (min) \ Win (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

C. 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 experimented 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 I 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 II 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

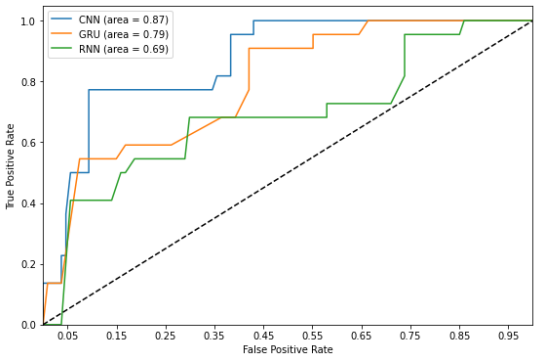

Fig. 6. ROC curve with CNN, RNN, GRU

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

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. 6 illustrates the receiver operating characteristic (ROC) curve for an accurate comparison of the three 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 and Future Work

In this work, we proposed a model that analyzes logs extracted from VMs which execute VNFs and determine whether failures will occur in the future. The proposed model is built by adapting the sentence classification techniques. To fit the log analysis of VMs, we made word embeddings with log corpus, and we used the pre-failure tagging method. We validate the proposed model with data generated in the OpenStack testbed.

The limitation of this research is that we used stress tools to generate failures instead of using real log data. So, now we are continuously collecting real log data and planning to use a large amount of data in the near future research.

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.

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] G. Xiao, Z. Zheng, B. Yin, K. S. Trivedi, X. Du and K. Cai, "Experience Report: failure Triggers in Linux Operating System: from Evolution Perspective," 2017 IEEE 28th International Symposium on Software Reliability Engineering (ISSRE), 2017, pp. 101-111.

[2] Q. Lin, K. Hsieh, Y. Dang, H. Zhang, K. Sui, Y. Xu, J. Lou, C. Li, Y. Wu and R. Yao, "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, 2018, pp. 480–490.

[3] H. Mi, H. Wang, Y. Zhou, M. R. Lyu, and H. Cai, "Toward fine-grained, unsupervised, scalable performance diagnosis for production cloud computing systems," IEEE Transactions on Parallel and Distributed Systems, 2013.

[4] M. Grottke and K. S. Trivedi, "A classification of software faults," Journal of Reliability Engineering Association of Japan, 2005, pp. 425-438.

[5] Q. Fu, J. Lou, Y. Wang and J. Li, "Execution Anomaly Detection in Distributed Systems through Unstructured Log Analysis," 2009 Ninth IEEE International Conference on Data Mining, 2009, pp. 149-158.

[6] S. He, J. Zhu, P. He and M. R. Lyu, "Experience Report: System Log Analysis for Anomaly Detection," 2016 IEEE 27th International Symposium on Software Reliability Engineering (ISSRE), Ottawa, ON, Canada, 2016, pp. 207-218.

[7] W. Ji, S. Duan, R. Chen, S. Wang and Q. Ling, "A CNN-based network failure prediction method with logs," 2018 Chinese Control And Decision Conference (CCDC), 2018, pp. 4087-4090.

[8] M. Du, F. Li, G. Zheng and V. 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, 2017, pp. 1285–1298.

[9] Y. Kim, "Convolutional neural networks for sentence classification," In Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP), 2014, pp. 1746–1751.

[10] D. Cotroneo, M. Grottke, R. Natella, R. Pietrantuono, and K. S. Trivedi, "Fault triggers in open-source software: An experience report," in Software Reliability Engineering (ISSRE), 2013 IEEE 24th International Symposium on. IEEE, 2013, pp. 178–187.

[11] T. Mikolov, I. Sutskever, K. Chen, G. Corrado and J. Dean, "Distributed Representations of Words and Phrases and their Compositionality", In Advances on Neural information Processing Systems, 2013.

[12] Adiscon GmbH, "The rocket-fast Syslog Server," [Online]. Available: https://www.rsyslog.com/ .

[13] Google, "word2vec," 2013. [Online]. Available: https://code.google.com/archive/p/word2vec/ .

[14] Y. Zhang and B.C. Wallace, "A sensitivity analysis of (and practitioners' guide to) convolutional neural networks for sentence classification," arXiv preprint arXiv:1509.01626 (2015).

[15] "Stress-ng," [Online]. Available: https://wiki.ubuntu.com/Kernel/Reference/stress-ng .

[16] The Apache Software Foundation, "Apache Bench," [Online]. Available: https://httpd.apache.org/docs/2.4/en/programs/ab.html .

[17] T.J. He, A.N. Toosi, and R. Buyya, "Performance evaluation of live virtual machine migration in SDN-enabled cloud data centers," Journal of Parallel and Distributed Computing 131 (2019): 55-68.