Detecting Anomalous Events on Distributed Systems Using Convolutional Neural Networks

Purimpat Cheansunan
Computer Engineering Department
King Mongkut's University of Technology Thonburi
Bangkok, Thailand
purimpat.che@gmail.com

Phond Phunchongharn
Computer Engineering Department
King Mongkut's University of Technology Thonburi
Bangkok, Thailand
phond.p@mail.kmutt.ac.th

Abstract— **Detection of anomalous events is very crucial for the maintenance and performance tuning in long-running distributed systems. System logs contain the complete information of system operation that can be used for describing the situations of the computing nodes. However, log messages are unstructured and difficult to utilize. In this work, we propose a novel anomaly detection framework in a Hadoop Distributed File System (HDFS) that transforms the log messages to structured data and automatically monitors the system operation logs using Convolutional Neural Networks (CNN). We evaluate the performance of anomaly detection in terms of precision, recall, and f-measure. The proposed framework can provide with precision = 94.76 ± 0.81%, recall = 99.53 ± 0.23%, and f-measure = 97.09 ± 0.49%. To apply the proposed framework in the practical application, we also concern about the training time and prediction productivity. From our experimental results, our proposed framework outperforms the existing models (i.e., LSTM and Bi-LSTM) with higher recall, lower training time, and higher prediction productivity.**

Keywords— Anomaly Detection, Deep Learning, Language model, Monitoring system, Preventive Maintenance

I. INTRODUCTION

Generally, a computer cluster composes of many computer servers requiring the high availability and efficient performance for computing the task. The unexpected failure of the machines seriously affects the computing tasks. Detecting an anomalous event as early as possible could assist the maintenance process in order to avoid an unexpected breakdown. Consequently, anomaly detection is essential for the distributed systems. In this work, we focus on a Hadoop Distributed File System (HDFS).

System operation logs fully contain the information about the system execution during runtime. The information can explain the events occurring in the system. System logs could be therefore used to detect the anomalous events which raise suspicions by differing significantly from the majority of the event in the system. In this work, we focus on anomaly detection though system operation logs.

Recently, there are several research papers attempt to apply deep learning techniques for detecting anomalous events [1-7]. Mostly, the family of Recurrent Neural Network (RNN) was chosen to handle the anomaly detection task. Du et al. proposed anomaly detection framework that can operate the detection on log-level called DeepLog [1]. The authors considered the sequence of log events as a structured language because it was produced by the structured source code written by developers. Thus, they aimed to build the language model on the normal execution log sequences using Long-Short Term Memory (LSTM). DeepLog outperformed the traditional methods as well as provided the automatically update the model to the unseen log entries. However, DeepLog is based on LSTM algorithm which is the recurrent network that unable to parallelize in the training process. Therefore, DeepLog spent so much time to build the model. Tuor et al also applied RNN to the cybersecurity task that aims to detect the anomalous authentication events in the system in [2-4]. The authors proposed several frameworks that applied several deep learning techniques including Deep Neural Network (DNN), and LSTM with various architectures. More specifically, they introduced the RNN attached with attention mechanism that produced the improvement on the model performance as well as produced more interpretable of the model while making a prediction. In [7], a CNN is applied to detect the anomaly event. Lu et al. came up with an anomaly detection framework with CNN. The authors developed a CNN that can extract the hidden relationship in logs by 2-dimension operation. As stated in the paper [7], experimental results showed that the model with CNN outperformed the one with LSTM since the characteristic of log sequence is not long enough to leverage the maximum performance of LSTM. Therefore, CNN is more suitable for detecting anomalies on the log sequence than LSTM. However, the authors performed the model training process as classification task which requires the output labels as either normal or abnormal. Additionally, the labeled dataset is difficult to obtain in the real world.

In this paper, we propose an anomaly detection framework through system log messages in an HDFS using a CNN. The proposed framework provides the practical approach to detect the anomalies in the real-time fashion. With the lower time consumption in a CNN, our proposed framework can provide with higher prediction productivity and lower training time against LSTM and Bi-LSTM framework. Furthermore, we evaluate our model on measurement metrics including Precision, Recall, and F-Measure. From our experimental results, we demonstrate that our proposed framework outperforms existing frameworks with the higher number of anomalies that model can capture as well as the higher prediction productivity with lower training time.

II. OUR PROPOSED ANOMALY DETECTION FRAMEWORK

This section describes the proposed framework, data preparation, anomaly detection model and performance evaluation.

A. *Our Proposed Framework*

To achieve our objective, we design our framework architecture which contains three primary parts which are data preparation, modeling, and evaluation as presented in

Fig. 1. Data preparation part is used to manipulate the log entries to be a suitable form for building the model. Then, the processed data (i.e., training and validating data sets) are passed to the modeling part in order to generate a deep learning model for detecting the anomalous behaviors of the system. Finally, the testing data set is fetched to evaluation part for measuring the model performance in detecting anomalies.

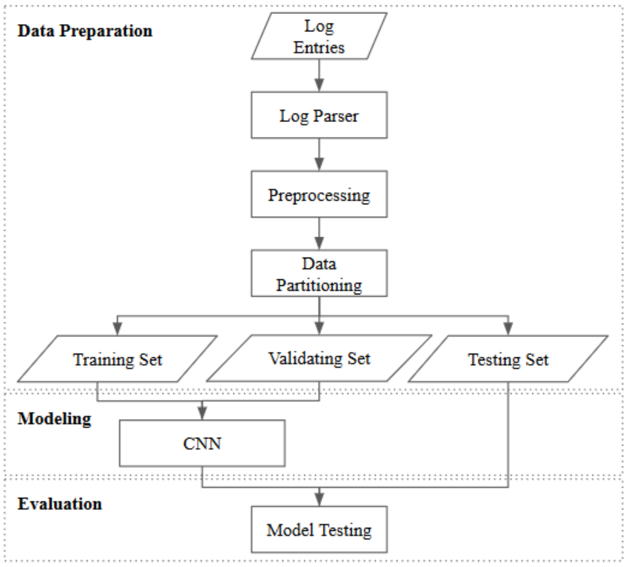
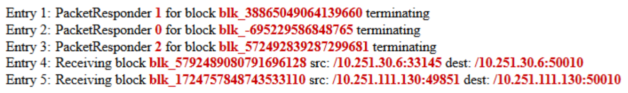


Fig. 1 Our Proposed Framework Architecture

B. Data Preparation

The system operation logs are usually collected in the unstructured form called log message. Most log messages are written by developers to produce some information about the system operation during system runtime that can be used to diagnosis the problems. Fig. 2 represents the example of HDFS log messages that compose of two components including constant component and variable component (bold). More specifically, the constant part is always the same. On the other hand, the variable part is changed by the value of a variable such as file name, directory, IP address, etc. We consider only the constant component of the message since it could be extracted and transformed from the unstructured form to the structured form, called as log keys. With the transformation, the machine could easily learn the structured data. In this part, we describe our data preparation process from raw log messages to processed data ready for modeling.



```
Entry 1: PacketResponder 1 for block blk_38865049064139660 terminating
Entry 2: PacketResponder 0 for block blk_-6952295868487 65 terminating
Entry 3: PacketResponder 2 for block blk_572492839287299681 terminating
Entry 4: Receiving block blk_5792489080791696128 src: /10.251.30.6:33145 dest: /10.251.30.6:50010
Entry 5: Receiving block blk_1724757848743533110 src: /10.251.111.130:49851 dest: /10.251.111.130:50010
```

Fig. 2 Example of HDFS log messages

1) Log Parser: is used to extract the log template which is the pattern of log message in order to differentiate between the constant and variable component in the log message. Log Parser could be developed by using several approaches, such as, Clustering, Frequent pattern mining, and Heuristics approach. The research in [8] has shown that Drain [9] log parser could achieve the highest performance in terms of parsing time usage and accuracy for HDFS. Drain applies Directed Acyclic Graph approach that consists of multiple layers in the architecture which provides the 60% faster than the other methods for mapping new log message with templates. Consequently, we use Drain as the log parser for obtaining log keys from raw log messages.

2) Preprocessing: is used to arrange outputs from the log parser with a timestamp and group them into a group of session by an identity field (i.e., block id in HDFS).

3) Data Partitioning: is used to separate the processed data into three sets including training set, validating set, and testing set. The training and validating sets are fetched into the modeling part for generating models and proving the accuracy of models, respectively. The training set entirely contains normal execution data which is used by the model to learn the pattern of normal behavior. Also, the validating set contains only normal event data which is used for validating the model to adjust parameters of the model in each iteration. On the other hand, testing set is passed to the evaluation part for checking the performance of models. The testing set therefore contains a mix of normal and abnormal behaviors. Specifically, the separation is done on session-level which is the group of log keys.

C. Anomaly Detection Model

In this paper, we introduced a novel convolutional architecture for the anomaly detection task. The architecture design is presented in Fig. 3. The model architecture is adapted from the existing research [7]. To make it more practical in the real world, we trained our model on the normal execution log events inspired by DeepLog [1] that aimed to build the language model. The model architecture composes of six layers including:

1) Input Layer: The sequence of log keys is fetched through this layer to the embedding layer.

2) Embedding Layer: The vector of log key sequences is encoded in this layer into the 2-dimensional matrix which is a suitable form for using in the convolution layer. Thus, the output shape from this layer is (Sequence length, Embedding size).

3) Convolution Layer (Conv1D): The convolution layer composes of six 1-dimensional convolutional components separated into two layers of three parallel components as shows in Fig. 3. Each of convolution pair has its own size of kernel and also differing to the another for considering different length of log key sequence (i.e. CNN with kernel size 3 considers three log keys locating to each other). Moreover, the output of convolution components are activated by Leaky Rectified Linear Unit (LReLU) for increasing the training speed as well as solving the dying ReLU problem. Lastly, we apply the Max pooling to gain only the maximum value of each output from LReLU for preventing the occurrence of overfitting problem.

4) Concatenation Layer: The outputs of convolution component are merged in the concatenation layer. Furthermore, we flatten the concatenated output into the vector which is the suitable form for fetching to the classifier layer.

5) Fully Connected Layer (FC): In this layer, the dropout regularizer is applied to prevent the overfitting. After that, the softmax activation produces the output.

6) Output Layer: The output of the model is the vector of the probability of log key to be the next log key in the sequence.

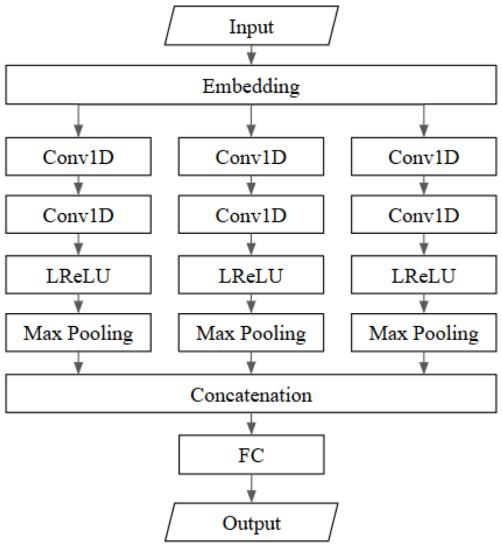


Fig. 3 Convolutional Neural Network Model Architecture with default parameter setting

Since our proposed model learns only the normal execution events, the model predicts only the normal log keys for the next sequence. If the actual log key differs from the predicted log keys, the proposed framework will identify as the anomalous event is detected.

D. Performance Evaluation

In this work, we treat the detection task as a classification task which consists of two classes (i.e., anomaly, and normal event). To evaluate the performance of our proposed framework, we use the following metrics: precision, recall, and f-measure presented as (1), (2), and (3) respectively. Note that positive means anomaly events and negative means normal events.

$$\text{Precision} = \frac{\text{True Positive (TP)}}{\text{True Positive (TP)+False Positive (FP)}} \quad (1)$$

$$\text{Recall} = \frac{\text{True Positive (TP)}}{\text{True Positive (TP)+False Negative (FN)}} \quad (2)$$

$$\text{F-measure} = \frac{2 \text{ x Precision x Recall}}{\text{Precision+Recall}} \quad (3)$$

For our detecting model, we focus on achieving less miss detection. Since the interpretation of recall is how well the model predicts the positive class (i.e., anomaly events), the recall must be highest as much as possible.

III. Analytical Results

For our performance evaluation, we employed the HDFS log dataset in [10]. The total number of sessions is 575,059 sessions. Since the data set is time-series, the data was separated in to three parts by periods of time as training, validating, and testing sets. Each session was labeled as either Normal or Abnormal. The descriptive of our data set is shown in Table I.

TABLE I. Dataset Description

Dataset	Number of sessions			Number of unique log keys
	Normal	*Anomaly*	*Total*	
Training	3,884	-	4,855	24
Validating	971	-		
Testing	553,366	16,838	570,204	29
Total	**558,221**	**16,838**	**575,059**	**29**

A. Parameter Tuning on Our Proposed Models

We examined on various CNN architecture designs and parameter setups in this experiment. Based on the model architecture presented in Fig. 3, we attempted to insert or remove some layers in order to determine the best architecture in anomaly detection. We defined the default parameter values including sequence length (w) = 10, number of convolution layers (cl) = 2, number of convolution components in parallel (n) = 3, kernel size (k) = (3, 4, 5), pooling size (p) = (8, 7, 6), number of candidate predictions (c) = 9. Note that, number of kernel size values depend on the number of convolution components because each component has its own kernel size differing to each other as described in II-C. As shown in Fig. 4, when c decreased, the recall increased but the precision dramatically decreased which cause by the ambiguous pattern of log sequence that make a model confuse.

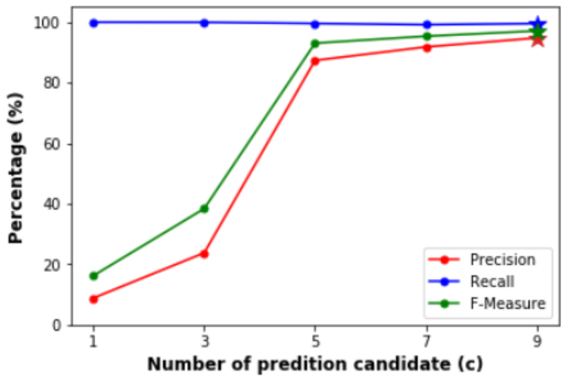


Fig. 4 The model performance when vary the number of prediction candidate (c)

Table II represents some examples of unclear input sequences among four log keys. We found similar patterns of log sequence that produce the different next log key. For example, the sequence **'5 11 9 11 9 11 9 26 26 26'** marked as bold character produces three log keys (i.e., 2, 3, and 4). From this point, the model could miss classifying a session as an anomaly since the ambiguous input sequence that causes the precision is low on c less than 5. Therefore, increasing c is the solution to solve the ambiguous pattern issue.

On the other way in Fig. 5, when n increased up to 3, a recall could be improved dramatically while precision moderately decreased. Moreover, when n was greater than

3, recall slightly decreased. From the experimental result, we realize that the information extracted from the sequence with length less than five is not cover some hidden relationship among the log events. However, the sequence with length more than five also provides some noise information to the model that reduce the model performance. Therefore, the proposed model with $c = 9$ and n = 3 could achieve the highest recall that provided the lowest miss detection.

TABLE II. SOME EXAMPLES OF AMBIGUOUS INPUT SEQUENCE

Input Sequence	Next log key
26 26 26 11 9 11 9 11 9 2	2
5 11 9 11 9 11 9 26 26 26	
5 26 26 26 11 9 11 9 11 9	
5 11 9 11 9 11 9 26 26 26	3
22 11 9 11 9 11 9 26 26 26	
5 11 9 11 9 11 9 26 26 26	
22 11 9 11 9 11 9 26 26 26	4
5 11 9 11 9 11 9 26 26 26	
5 11 9 11 9 26 26 11 9 26	
26 26 26 11 9 11 9 11 9 2	23
5 26 26 26 11 9 11 9 11 9	
11 9 26 26 11 9 26 3 4 3	

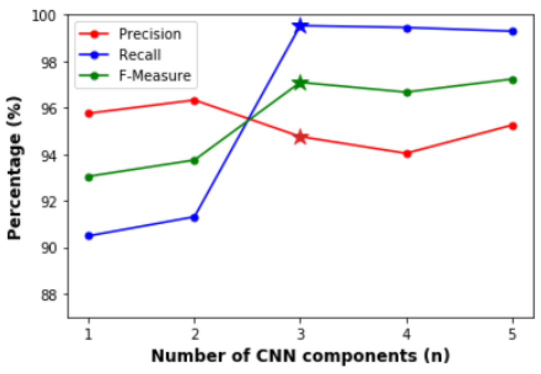


Fig. 5 The model performance when vary the number of convolutional components (n)

IV. COMPARISON AND DISCUSSION

We also compare our proposed model with the baseline model presented in the DeepLog framework [1] which is the state-of-the-art anomaly detection framework at the present. The models were implemented using Long-Short-Term-Memory (LSTM) which is the family of recurrent network. Moreover, we also compare our proposed model with the advance design of LSTM called Bidirectional LSTM (Bi-LSTM) which was applied to build language model by Tuor et al. [3]. For each model, we measured three performance metrics as described in section II-D. This experiment was set up to compare our proposed model with the baseline models. For both baseline models, we set up model parameters follow the setting that stated in [1] in order to obtain the baseline detection efficiency. The parameter settings are input sequence length (w) = 10, memory size (m) = 64, number of LSTM layers (l) = 2, and number of prediction candidates (c) = 9. Furthermore, we evaluate the efficiency of the embedding layer, which encode the log key sequences into 2-dimensional matrices. For fair comparison, we added the embedding layer to both LSTM and Bi-LSTM.

A. *Comparing on Anomaly Detection Performance*

From experimental results as shown in Table III, Bi-LSTM could outperform the other models in terms of recall and F-measure when without the embedding layer. When we applied the embedding layer, it could improve recall significantly. However, precision in both LSTM and Bi-LSTM models could not be improved while that in our proposed model (CNN) could be remarkably improved about 20 percent. The embedding layer provides more information to the model via latent features extracted from the log key that improve the predicting efficiency. Moreover, CNN with embedding represented as bold character could provide the higher recall and F-Measure comparing to LSTM and Bi-LSTM models with the embedding layer because CNN is better to find a local pattern in the sequence.

TABLE III. COMPARING WITH LSTM AND BI-LSTM MODELS

Approach	Precision (%)	Recall (%)	F-Measure (%)
LSTM + No Embedding	96.25	93.78	94.99
LSTM + Embedding	96.46	95.44	95.94
Bi-LSTM + No Embedding	95.79	96.84	96.31
Bi-LSTM + Embedding	95.21	97.39	96.28
CNN + No Embedding	74.89	93.68	83.23
CNN + Embedding (cl = 1)	**94.44**	**98.90**	**96.62**
CNN + Embedding (cl = 2)	**94.76**	**99.53**	**97.09**

B. *Comparing on Time Consumption*

To make our proposed framework more practical, we consider the performance in term of time-consuming such as training time as well as the prediction productivity of the model. Training time is the time spent during modeling detection model (i.e., modeling CNN). We display the time-consuming on the training process among LSTM, Bi-LSTM and our proposed model (CNN) in Fig. 6. The figure represents the training time of both without and with the embedding layer. Additionally, the information of our computing environment that we performed our experiments is shown in Table IV.

TABLE IV. COMPUTING ENVIRONMENT

Component	Description
Operating System	Ubuntu 16.04 LTS
CPU	Intel Broadwell 8 cores
Ram	32 GB
Hard disk	100 GB

As shown in Fig. 6, the model with the embedding layer consumed time to build the model higher than the model without embedding. This is the tradeoff in higher recall and F-score as stated in Table II. Moreover, the training time spent in LSTM and Bi-LSTM higher than our proposed model about 3 and 3.5 times, respectively because CNN operations are fully parallelized without depending on the other time step as normally done in the RNNs architectures (specifically, LSTM and Bi-LSTM) which spent more time in training process.

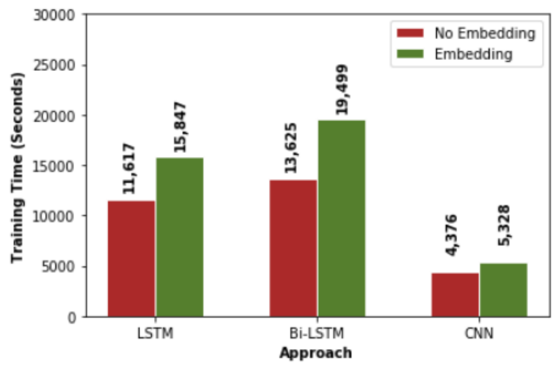


Fig. 6 Training time comparison

Furthermore, we concern about the predicting productivity in terms of the number of the log event that model can predict per second. As shown in Fig. 7, our proposed models provided the higher productivity than both LSTM and Bi-LSTM about 2.5 times. As described above, CNN performs the parallelized computation, therefore, the model can make a prediction faster than the RNNs family which operated in sequential.

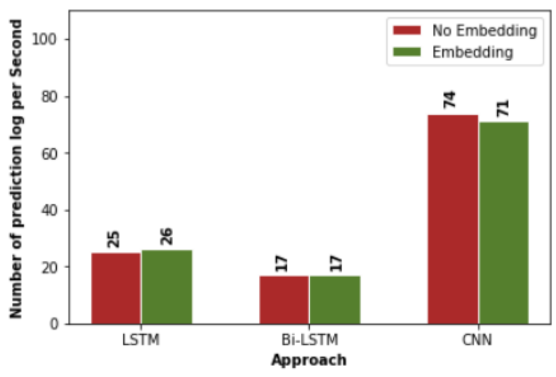


Fig. 7 Number of prediction log per second

In the practical situation, anomaly detection must be able to support the real-time system. Hence, the detection model must have low training time and high prediction productivity. Therefore, our proposed framework is more practical to the real-world applications compared to LSTM and Bi-LSTM models which are the popular model architectures in anomaly detection field.

V. Conclusion

In this paper, we have introduced a novel anomaly detection frame using the deep learning technique, called convolutional neural network (CNN) to detect the anomalous events on system operation logs in Hadoop Distributed File System (HDFS). The proposed model (CNN) is applied to build a language model on the normal execution log events for predicting the most possible event (log key) according to the incoming sequence of events. The anomalous event will be detected when the actual log event deviates from the prediction of the model. We evaluated the performance of our proposed framework in terms of precision, recall, and F-measure. Due to anomaly detection, we have focused on recall performance in order to avoid the miss detection. From the experimental results, our proposed framework could outperform the existing models in anomaly detection field including both LSTM and Bi-LSTM models with higher recall and F-measure. According to experimental results, we have identified the best parameter setting for our proposed CNN model that could achieve the precision = 94.76 ± 0.81%, recall = 99.53 ± 0.23%, and f-measure = 97.09 ± 0.49%. Moreover, CNN can provide more productive prediction than LSTM and Bi-LSTM because the CNN operations are fully parallelized without any depending on other time-step that make our proposed model consume lower training and predicting time than RNNs architecture.

Furthermore, we plan our future work to apply the attention mechanism, popular in translation task, to our model for extracting input sequence and predicting output target as a sequence of log keys instead of only single log key.

Acknowledgment

This paper was supported by the Big Data Experience Center, King Mongkut's University of Technology Thonburi (KMUTT), Thailand.

References

[1] 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, Texas, 2017, pp. 1285-1298.

[2] A. Tuor, S. Kaplan, B. Hutchinson, N. Nichols, and S. Robinson, "Deep learning for unsupervised insider threat detection in structured cybersecurity data streams," In Workshops at the Thirty-First AAAI Conference on Artificial Intelligence, 2017.

[3] AR. Tuor, R. Baerwolf, N. Knowles, B. Hutchinson, N. Nichols, and R. Jasper, "Recurrent neural network language models for open vocabulary event-level cyber anomaly detection," In Workshops at the Thirty-Second AAAI Conference on Artificial Intelligence, 2018.

[4] A. Brown, A. Tuor, B. Hutchinson, and N. Nichols, "Recurrent neural network attention mechanisms for interpretable system log anomaly detection," In Proceedings of the First Workshop on Machine Learning for Computing Systems, 2018, p. 1.

[5] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, AN. Gomez, Ł. Kaiser, and I. Polosukhin, "Attention is all you need," In Advances in Neural Information Processing Systems 2017, California, 2017, pp. 5998-6008.

[6] M. Elbayad, L. Besacier, and J. Verbeek, "Pervasive attention: 2D convolutional neural networks for sequence-to-sequence prediction," In CoNLL 2018-Conference on Computational Natural Language Learning, Brussels, 2018, pp. 1-11.

[7] S. Lu, X. Wei, Y. Li, and L. Wang, "Detecting anomaly in big data system logs using convolutional neural network," In 2018 IEEE 16th Intl Conf on Dependable, Autonomic and Secure Computing, 16th Intl Conf on Pervasive Intelligence and Computing, 4th Intl Conf on Big Data Intelligence and Computing and Cyber Science and Technology Congress (DASC/PiCom/DataCom/CyberSciTech), Athens, 2018, pp. 151-158.

[8] J. Zhu, S. He, J. Liu, P. He, Q. Xie, Z. Zheng, and MR. Lyu, "Tools and benchmarks for automated log parsing," In Proceedings of the 41st International Conference on Software Engineering: Software Engineering in Practice, 2019, pp. 121-130.

[9] P. He, J. Zhu, Z. Zheng, and MR. Lyu, "Drain: an online log parsing approach with fixed depth tree," In 2017 IEEE International Conference on Web Services (ICWS), 2017, pp. 33-40.

[10] W. Xu, L. Huang, A. Fox, D. Patterson, and MI. Jordan, "Detecting large-scale system problems by mining console logs," In Proceedings of the ACM SIGOPS 22nd Symposium on Operating Systems Principles, Montana, 2009, pp. 117-132.

[11] W. Xu, L. Huang, A. Fox, D. Patterson, and M. Jordan, "Online system problem detection by mining patterns of console logs," In Ninth IEEE International Conference on Data Mining, Florida, 2009, pp. 588-597.