Improving Log-Based Anomaly Detection with Component-Aware Analysis

Kun Yin[1,2], Meng Yan[1,2,*], Ling Xu[1,2], Zhou Xu[1,2], Zhao Li[1,2], Dan Yang[2] and Xiaohong Zhang[1,2]
[1]Key Laboratory of Dependable Service Computing in Cyber Physical Society (Chongqing University), Ministry of Education, China
[2]School of Big Data & Software Engineering, Chongqing University, Chongqing, China
Email: kunyinzero@gmail.com, {mengy, xuling, zhouxullx, lizhao, dyang, xhongz}@cqu.edu.cn

***Abstract*—Logs are universally available in software systems for troubleshooting. They record system run-time states and messages of system activities. Log analysis is an effective way to diagnosis system exceptions, but it will take a long time for engineers to locate anomalies accurately through logs. Many automatic approaches have been proposed for log-based anomaly detection. However, most of the prior approaches did not consider the corresponding system component of a log message. Such component records the log location, which can help detect the location-sequence-related anomalies. In this paper, we propose LogC, a new Log-based anomaly detection approach with Component-aware analysis. LogC contains two phases: (i) turning log messages into log template sequences and component sequences, (ii) feeding such two sequences to train a combined LSTM model for detecting anomalous logs. LogC only needs normal log sequences to train the combined model. We evaluate LogC on two open-source log datasets: HDFS and ThunderBird. Experimental results show that LogC overall outperforms three baselines (i.e., PCA, IM, and DeepLog) in terms of three metrics (precision, recall, and F-measure).**

***Index Terms*—Anomaly Detection, Log Analysis, Deep Learning**

I. Introduction

With software systems evolving to large-scale and complex distributed systems, these systems often suffer from bugs and vulnerabilities. Besides, these large-scale systems often provide massive online services and application program interfaces, which require high robustness and stability. But when a system failure occurs (such as service fault and service outage), multiple services may be affected by the failure, which may lead to a significant loss of the system. The system anomaly detection technique aims to locate these system failures. Such a technique plays a vital role in system maintenance. Timely and precise anomaly detection is necessary for engineers to pinpoint causes promptly.

There are many types of data for anomaly detection and troubleshooting in the system. Log data, which is universally available in most of the large-scale systems, have a wealth of information and contain a record of critical system states, events, and run-time messages. So system logs become a central data source for anomaly detection. Log-based anomaly detection has become a research question, and many approaches [1]–[3] have been proposed. Generally, these existing approaches first extract useful features from logs and then employ unique or universal detection algorithm to locate anomalous logs. In the process, log feature selection is vital because the representative features can reflect the difference between regular logs and abnormal logs. However, prior approaches did not consider components in logs when building a log-based anomaly detection model. Components record the location of logs and reflect the calling relationship between system modules. Some system failures may result in different component workflows that can be reflected in logs, but they may hard to be detected through log template sequences.

In this paper, we propose a new method, called LogC, which utilizes component-aware analysis, to pinpoint abnormal log sequences. Component-aware analysis is a mechanism that integrates the detection of abnormal component workflows. The proposed method LogC first transforms unstructured log data to log template sequences and component sequences. Then, LogC trains two LSTM models from normal log template sequences and component sequences, respectively. Finally, LogC combines two LSTM models for anomaly detection. We evaluate LogC over two log datasets following prior studies [3], [4], and the results show the effectiveness of our method.

The main contributions of this paper are as follows:

- We propose a new log-based anomaly detection with component-aware analysis: LogC. Experimental results on two widely used log datasets show that LogC achieves an F-measure of 0.949 on average, which outperforms three baseline approaches.
- We investigate how LogC performs under different configurations. Experimental results show that the threshold, which determines the anomaly detection standard of LogC, has a significant impact on the effectiveness of LogC.
- The proposed component-aware analysis can be adapted to other existing log-based anomaly approaches.

II. The Proposed Approach

Overview. LogC aims to automatically and accurately detect anomalous logs that reflected system failures. System logs, which are printed by logging statements, can be separated into several classes by their contents [5]. Each log contains its component information that means which system modules the log message belongs to. Our main idea is to improve the

*Corresponding author.

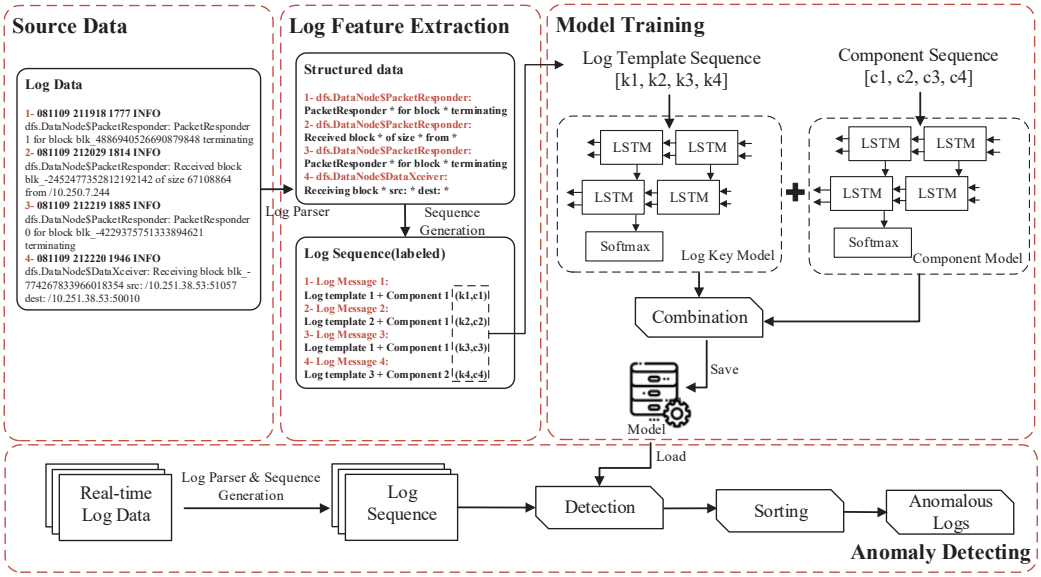


Fig. 1. The framework of LogC

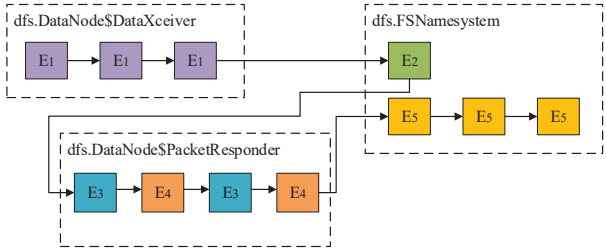


Fig. 2. An example of an HDFS block

effectiveness of log-based anomaly detection through a combined LSTM model considering component-aware analysis. The architecture of our approach is shown in Figure 1. LogC first extract templates and components from historical logs, and then generate log sequences and component sequences. Based on the combined LSTM model, LogC can identify abnormal logs once a new log sequence and component sequence are fed into the trained model.

To better demonstrate component-aware analysis, Figure 2 shows the workflow of an HDFS block (identified by block IDs) containing several log messages, and the change of component. In Figure 2, E_t indicates a class of log templates. *"dfs.FSNamesystem"* is a system component of the Hadoop Distributed File System (HDFS). Some system failures may result in different component workflows. These system failures could be located by monitoring changes in the component sequences. Thus we propose a combined model that can process two types of sequences at the same time.

Log Feature Extraction. Logs are unstructured and contain a wealth of information with free format. The purpose of log parsing is to extract log keys, also known as log templates, from log messages. To be more specific, every message can be parsed into a kind of template with some parameters. For example, according to Figure 1, we can extract block IDs and IP address from HDFS logs.

In our approach, Drain [6] is applied to build log template sequences from log data. Besides, we use *regular expressions* to match components and HDFS block IDs.

Model Training. Suppose a system log file contains n log templates $L = \{l_1, l_2, ..., l_n\}$ and m components $O = \{o_1, o_2, ..., o_m\}$. Obviously, $n \geq m$. Let m_t denotes the log message of time t. For example, giving a log message sequence $M = \{m_1, m_2, m_3, ..., m_t\}$, through log feature extraction, a new sequence $M_e = \{(c_1, k_1), (c_2, k_2), (c_3, k_3), ..., (c_t, k_t)\}$ is generated. c_t is a component of m_t at time t. k_t is a log template of m_t at time t. The input of the combined LSTM model is a window W of the h most recent log templates and components. $W = \{(c_{t-h}, k_{t-h}), (c_{t-h+1}, k_{t-h+1}), ..., (c_{t-1}, k_{t-1})\}$, where each pair (c_t, k_t) is extracted from log message m_t. For example, suppose a log sequence $M_e = \{(o_1, l_6), (o_1, l_4), (o_4, l_7), (o_3, l_{22})\}$, window size $h = 2$, then the training data are $\{(o_1, l_6), (o_1, l_4) \rightarrow (o_4, l_7)\}$, $\{(o_1, l_4), (o_4, l_7) \rightarrow (o_3, l_{22})\}$.

The combined model is a multi-class classifier, so the loss function in training is cross-entropy. For the problem, cross-entropy is used as the loss function for log key model and component model, which is shown as follows:

$$loss = -\frac{1}{M}\sum_{i=1}^{M}\sum_{j=1}^{n} k_j log(p_j) \quad (1)$$

M is the total number of training samples and n is the number of types of log templates or components. k_j is the real tag of the log template or the component. p_j is the probability value of the log key j or the system component j. Adam algorithm is applied to minimize the loss function.

Anomaly Detecting. LogC can run online. To confirm if a log message m_t is normal or abnormal, we send window $M_h = \{m_{t-h}, m_{t-h+1}, ..., m_{t-1}\}$ to LogC. Firstly, these log messages will be extracted to form structured data as the input of the trained combined model. And the corresponding output are two probability distributions $P_l[l_t|W] = \{l_1 : p_{l_1}, l_2 :$

TABLE I
DETAILS OF THE DATASETS

Datasets	Original logs	Anomalies	Anomalies(%)
HDFS	11,175,629	16838(blocks)	2.93
ThunderBird	3,992,351	162,953	4.08

$p_{l_2}, ..., l_n : p_{l_n}\}$ and $P_o[o_t|W] = \{o_1 : p_{o_1}, o_2 : p_{o_2}, ..., o_m : p_{o_m}\}$. Then our approach sorts the possible log templates L and components O by the probabilities that are the outputs of the *softmax* activation function.

If the real log template k_t and component c_t are not among the top g log templates L_g and components O_g with relatively large probabilities, the real message m_t will be treated as an anomalous log. And an alarm of a system failure will be generated. Specifically, there is a calculation process to determine whether the log message m_t is anomalous. We define that r_l is the prediction result of the log template of m_t, and r_o is that of the component of m_t. The final result $r = r_o \ \boldsymbol{and}\ r_l$. There is a special case that occurs when $r_o = 1$ and $r_l = 0$. The corresponding result $r = 1$. If $r = 1$, the log message m_t will be labeled as an anomalous log message.

III. Experimental Setup

Research Questions. We design experiments to answer the following research questions:

RQ1: *How effective is LogC in detecting anomalous logs?*

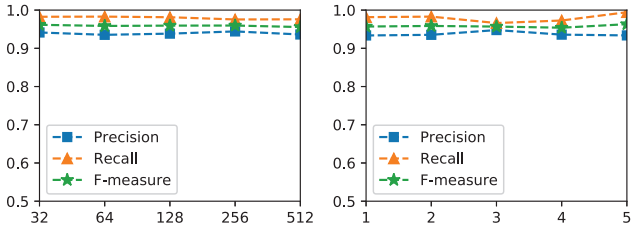
RQ2: *How does LogC perform under different configurations?*

RQ3: *How does the proposed component-aware analysis perform by integrating other anomaly detection approaches?*

Datasets and Setup. We use two open-source datasets to evaluate our approach following prior studies [3], [4]. The basic information of the datasets are reported in Table I. These logs are from a distributed system and a high-performance computer system. HDFS data are collected from more than 200 Amazon's EC2 nodes. The dataset has been labeled by experts. ThunderBird is an open-source log dataset collected from a ThunderBird supercomputer system at Sandia National Labs. It has more than two hundred million logs, and we choose the top about four million log messages. The log contains alert and non-alert messages identified by tags.

Preprocessing is needed to cut these long log sequences, which are generated by large systems, into small ones [7]. In the following experiments, on HDFS, We group log messages into different sessions by block IDs and each session is a life cycle of a block. There are 575,061 blocks in HDFS. We leverage top about 1% normal logs as the training data (if needed), and the rest as the testing data (both abnormal and normal logs). On ThunderBird, we divide the dataset by fixed windows because ThunderBird logs have no obvious identifier. We label a session as abnormal if there is any anomalous log in the session. Top 80% of the logs are selected as the training data, and the rest are the testing data.

Baselines. We compare our approach with three baseline methods. The detecting process of these methods are analogous. These detection methods include: PCA [3], IM [2], DeepLog [1]. We implement our approach and DeepLog with Python 3.6 and PyTorch 1.2.0. As for PCA and IM, we use an open-source toolkit provided by He et al. [5].



(a) Memory units: α_k (b) Layers: L_k

Fig. 3. LogC performance with different α_k and L_k on HDFS

The *Principal Component Analysis (PCA)* groups log messages by special identifiers. Utilizing a log parsing method and counter, it can get log template vectors from log sessions. PCA detects an abnormal vector by calculating the projected length on anomalous space.

Invariant Mining (IM) groups log messages according to the relationship among log parameters. The approach can reveal the inherent linear characteristics of event workflows by mined invariants.

DeepLog is a state-of-the-art anomaly detection proposed by Du et al. [1]. The approach performs well on multiple datasets: HDFS, OpenStack and BGL. DeepLog aims to detect sequential anomalies and parameter anomalies of log data utilizing LSTM networks.

Evaluation Metrics. *FP* is the number of normal logs (blocks) that are identified as abnormal logs (blocks). *FN* means how many abnormal logs (blocks) are identified as normal logs. *TP* denotes the number of anomalous logs (blocks) that are correctly identified. Anomaly detection is a binary classification problem, so precision, recall and F-measure are usually used as assessment criteria. Precision=$\frac{TP}{TP+FP}$, recall=$\frac{TP}{TP+FN}$, F-measure=$\frac{2 \cdot Precision \cdot Recall}{Precision+Recall}$.

IV. Results

In this section, we present our experimental results by answering the research questions mentioned in Section III.

A. RQ1: How effective is LogC in detecting anomalous logs?

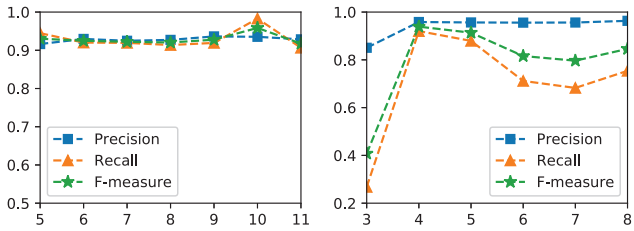
Table II shows the results achieved by LogC. We evaluate three baselines (i.e., PCA, IM, and DeepLog) and our approach on two log datasets: HDFS and ThunderBird.

By default, we set $g_k = 9$, $h_k = 10$, $L_k = 2$, $\alpha_k = 64$ for the log key model [1] and $g_c = 6$, $h_c = 10$, $L_c = 1$, $\alpha_c = 64$ for the component model. g determines the anomaly detection standard of LogC. h denotes the window size. L is the number of LSTM network layers and α denotes memory units of one LSTM cell. The value of h_k and h_c must be equal.

As shown in Table II, our approach achieves a recall of 98.29% and an F-measure of 95.85% over the HDFS dataset. PCA has the highest precision of 97.73% but at the cost of a lower recall, which means that it achieves more false negatives. In a large-scale system, for a log-based detection method, low recall indicates that the monitoring ability of the detection

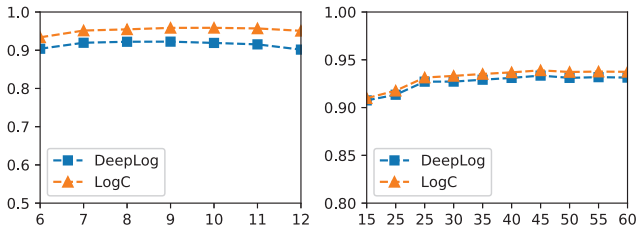
TABLE II
RESULTS ON THE TWO DATASETS

Methods	HDFS			ThunderBird		
	Precision(%)	Recall(%)	F-measure(%)	Precision(%)	Recall(%)	F-measure(%)
PCA	**97.73**	74.71	84.68	63.16	62.80	62.96
IM	73.22	**100.00**	84.54	59.18	90.34	71.52
DeepLog	93.37	91.15	92.25	**96.40**	90.47	93.34
LogC	93.53	98.29	**95.85**	95.83	**92.03**	**93.89**



(a) HDFS (b) ThunderBird

Fig. 4. The impact of h on the effectiveness of LogC



(a) HDFS (b) ThunderBird

Fig. 5. The F-measure of two approaches with different g, g_k

module is feeble though it takes up specific system resources. It is inefficient.

On ThunderBird dataset, there are more log templates and components than that of HDFS. So we set $g_k = 45$, $g_c = 15$. And we reduce the size of window W by setting $h = 4$ because of the nature of ThunderBird logs. As illustrated in Table II, Deeplog decreases F-measure by 0.55% and recall by 1.56% compared to LogC.

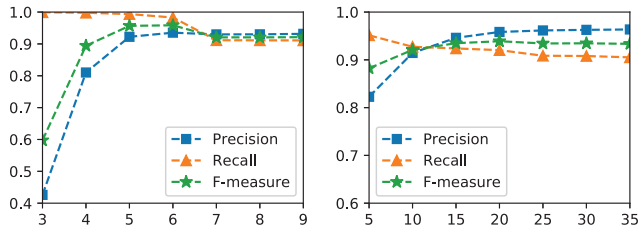
In summary, our approach LogC achieves the best results on the two log datasets in terms of F-measure. LogC has higher values of recall compared to PCA and DeepLog, which means LogC achieves less false negatives.

B. RQ2: How does LogC perform under different configurations?

As discussed in Section II, LogC utilizes LSTM to learn models of two kinds of sequences and uses a parameter $g(g_k, g_c)$ as the threshold. So we investigate the impact of some critical parameters in LogC from two aspects. In each experiment, we change the value of one parameter and fix the values of the others to the default values as shown in *RQ1*.

Parameters in LSTM Network. These hyperparameters contain: memory units α_k, layers L_k, and window size h. As can be seen from Figure 3, varying the value of L_k and α_k will not affect the performance of LogC much on HDFS. It is the same on ThunderBird.

According to Figure 4(b), window size h has a significant impact on the performance of LogC over ThunderBird. Further



(a) HDFS (b) ThunderBird

Fig. 6. The impact of g_c on the effectiveness of LogC

TABLE III
RESULTS OF NEW METHODS ON HDFS AND THUNDERBIRD

Methods	Datasets	Precision(%)	Recall(%)	F-measure(%)
PCA-C	HDFS	96.52	**91.27**	**93.82**
	ThunderBird	**64.99**	**77.60**	**70.76**
IM-C	HDFS	72.70	100.00	84.19
	ThunderBird	**60.03**	**93.60**	**73.15**

research shows that the nature of the two types of logs is different. If we adjust the value of g_k and g_c shown in *RQ1*, LogC can achieves an F-measure of 86.56%. So threshold g (g_k, g_c) plays a vitally important role in this case.

The Threshold in the Detection Stage. We investigate the impact of g (g_k, g_c) on LogC over two datasets. According to Figures 5(a) and 5(b), LogC performs better overall than DeepLog in terms of F-measure. Threshold g in DeepLog and threshold g_k in LogC have the same meaning. In Figure 6(a), F-measure changes dramatically because there only are few components in HDFS logs. When we set $g_c = 3$, the precision of LogC is very low, which means a large number of normal logs are identified as abnormal logs.

In conclusion, the threshold g_c has a significant impact on the effectiveness of LogC. Other parameters in LogC, have little effect on LogC.

C. RQ3: How does the proposed component-aware analysis perform by integrating other anomaly detection approaches?

To investigate the importance of component-aware analysis and the effectiveness of the component model in LogC, we combine the baselines (PCA and IM) with the component model. The new methods are named PCA-C and IM-C. The detection model of LogC is composed of DeepLog and component model. We replace the log key model with the two anomaly detection approaches [2], [3]. For each log session, baselines and the component model give the prediction values, respectively. And through the calculation mentioned in Section II, the new methods output the final results.

We evaluate the two new approaches over the HDFS dataset and ThunderBird dataset. As shown in Table III, compared with the performance of baselines in Table II, the new methods have higher values of recall and F-measure and perform better.

However, we find that the results of IM-C on HDFS are not as good as the previous method IM. Further research shows that combining the component model is an effective way to improve the recall of a detection method, but this combination may slightly reduce the precision of the method. In this case, IM achieves a recall of 100% on HDFS, so the recall of IM cannot be improved. The results of PCA-C and IM-C over ThunderBird verify the truth of our findings and the effectiveness of component-aware analysis.

In summary, through integrating component-aware analysis, baselines (i.e., PCA and IM) perform better in most cases. However, LogC still outperforms the two new approaches (i.e., PCA-C and IM-C).

V. Related Work

Log Parsing. Logs play an essential role in the development and maintenance of large-scale systems. In recent years, many efficient approaches have been proposed to parse system log data based on frequent pattern mining and clustering. For LenMa [8] and LogMine [9] methods, the clustering algorithm was applied to parser system logs. Spell [10] was an online streaming method that parsed system event logs through the longest common subsequence. Drain method [6] utilized a fixed depth parse tree to get structured data. In addition, some industrial log management tools also have high efficiency with powerful search technology and intelligent algorithm. Our work applies Drain to parse logs because it performs well on multiple log datasets [11].

Log-Based Anomaly Detection. A large number of detection methods were designed for specific systems, and many studies so far has focused on log-based anomaly detection. He et al. [5] provided a detailed review of log-based anomaly detection. Usually, for anomaly detection approaches that view a log file as a log sequence, it is essential to extract sequential log information accurately. These approaches can be basically classified into two categories: supervised anomaly detection and unsupervised anomaly detection. Labeled training data is the prerequisite of supervised methods [12], [13].

However, unlike supervised methods, unsupervised methods do not need training data, so they are more suitable for a real-world system environment [2], [3], [14]. In recent years, the recurrent neural network (RNN) is widespread and achieves excellent performance in sequence prediction. Researchers had applied multiple types of RNN on log sequence anomaly detection [1], [15], [16]. Our work also utilizes the LSTM model but considers more features of logs.

There also exist other special anomaly detection methods. Some approaches focused on workflow construction [17]–[20]. In addition, probabilistic analysis was also used to detect anomalies through log data [21], [22].

VI. Conclusion and Future work

In this paper, we proposed LogC, a new log-based anomaly detection approach combining component-aware analysis, to help engineers to pinpoint anomalous logs. LogC first transforms unstructured log data to log template sequences and component sequences. With such two sequences, LogC trains two LSTM models and combines them for anomaly detection. Experimental results show that LogC outperforms three baselines (i.e., PCA, IM, and DeepLog) in terms of F-measure. In the future, we will choose more features in log data and apply Attention Mechanism to log sequence prediction.

Acknowledgement. This work was supported in part by the National Key Research and Development Project (No.2018YFB2101200), the Fundamental Research Funds for the Central Universities (No.2019CDYGYB014 and No.2020CDJQY-A021).

References

[1] M. Du, F. Li, G. Zheng, and V. Srikumar, "Deeplog: Anomaly detection and diagnosis from system logs through deep learning," in *CCS*, 2017, pp. 1285–1298.

[2] J.-G. Lou, Q. Fu, S. Yang, Y. Xu, and J. Li, "Mining invariants from console logs for system problem detection." in *USENIX*, 2010, pp. 1–14.

[3] W. Xu, L. Huang, A. Fox, D. Patterson, and M. I. Jordan, "Detecting large-scale system problems by mining console logs," in *SOSP*, 2009, pp. 117–132.

[4] A. Oliner and J. Stearley, "What supercomputers say: A study of five system logs," in *DSN*. IEEE, 2007, pp. 575–584.

[5] S. He, J. Zhu, P. He, and M. R. Lyu, "Experience report: System log analysis for anomaly detection," in *ISSRE*. IEEE, 2016, pp. 207–218.

[6] P. He, J. Zhu, Z. Zheng, and M. R. Lyu, "Drain: An online log parsing approach with fixed depth tree," in *ICWS*. IEEE, 2017, pp. 33–40.

[7] J. Chen, W. Shang, A. E. Hassan, Y. Wang, and J. Lin, "An experience report of generating load tests using log-recovered workloads at varying granularities of user behaviour," in *ASE*. IEEE, 2019, pp. 669–681.

[8] K. Shima, "Length matters: Clustering system log messages using length of words," *arXiv preprint arXiv:1611.03213*, 2016.

[9] H. Hamooni, B. Debnath, J. Xu, H. Zhang, G. Jiang, and A. Mueen, "Logmine: Fast pattern recognition for log analytics," in *CIKM*, 2016, pp. 1573–1582.

[10] M. Du and F. Li, "Spell: Online streaming parsing of large unstructured system logs," *TKDE*, vol. 31, no. 11, pp. 2213–2227, 2018.

[11] J. Zhu, S. He, J. Liu, P. He, Q. Xie, Z. Zheng, and M. R. Lyu, "Tools and benchmarks for automated log parsing," in *ICSE-SEIP*. IEEE, 2019, pp. 121–130.

[12] Y. Liang, Y. Zhang, H. Xiong, and R. Sahoo, "Failure prediction in ibm bluegene/l event logs," in *ICDM*. IEEE, 2007, pp. 583–588.

[13] M. Chen, A. X. Zheng, J. Lloyd, M. I. Jordan, and E. Brewer, "Failure diagnosis using decision trees," in *ICAC*. IEEE, 2004, pp. 36–43.

[14] Q. Lin, H. Zhang, J.-G. Lou, Y. Zhang, and X. Chen, "Log clustering based problem identification for online service systems," in *ICSE-C*. IEEE, 2016, pp. 102–111.

[15] W. Meng, Y. Liu, Y. Zhu, S. Zhang, D. Pei, Y. Liu, Y. Chen, R. Zhang, S. Tao, P. Sun *et al.*, "Loganomaly: Unsupervised detection of sequential and quantitative anomalies in unstructured logs," in *IJCAI*, vol. 7, 2019, pp. 4739–4745.

[16] X. Zhang, Y. Xu, Q. Lin, B. Qiao, H. Zhang, Y. Dang, C. Xie, X. Yang, Q. Cheng, Z. Li *et al.*, "Robust log-based anomaly detection on unstable log data," in *ESEC/FSE*, 2019, pp. 807–817.

[17] A. J. Oliner, A. Aiken, and J. Stearley, "Alert detection in system logs," in *ICDM*. IEEE, 2008, pp. 959–964.

[18] A. Nandi, A. Mandal, S. Atreja, G. B. Dasgupta, and S. Bhattacharya, "Anomaly detection using program control flow graph mining from execution logs," in *KDD*, 2016, pp. 215–224.

[19] X. Yu, P. Joshi, J. Xu, G. Jin, H. Zhang, and G. Jiang, "Cloudseer: Workflow monitoring of cloud infrastructures via interleaved logs," *ACM SIGARCH Computer Architecture News*, vol. 44, no. 2, pp. 489–502, 2016.

[20] Q. Fu, J.-G. Lou, Q. Lin, R. Ding, D. Zhang, and T. Xie, "Contextual analysis of program logs for understanding system behaviors," in *MSR*. IEEE, 2013, pp. 397–400.

[21] S. Du and J. Cao, "Behavioral anomaly detection approach based on log monitoring," in *BESC*. IEEE, 2015, pp. 188–194.

[22] A. J. Oliner, A. V. Kulkarni, and A. Aiken, "Using correlated surprise to infer shared influence," in *DSN*. IEEE, 2010, pp. 191–200.