LogStamp: Automatic Online Log Parsing Based on Sequence Labelling

Shimin Tao, Weibin Meng*
Huawei
China

Yimeng Chen
Huawei
China

Yichen Zhu
University of Toronto
Canada

Ying Liu
Tsinghua University
China

Chunning Du
Beijing University of Posts and Telecommunications
China

Tao Han, Yongpeng Zhao, Xiangguang Wang, Hao Yang
Huawei
China

ABSTRACT

Logs are one of the most critical data for service management. It contains rich runtime information for both services and users. Since size of logs are often enormous in size and have free handwritten constructions, a typical log-based analysis needs to parse logs into structured format first. However, we observe that most existing log parsing methods cannot parse logs online, which is essential for online services. In this paper, we present an automatic online log parsing method, name as *LogStamp*. We extensively evaluate LogStamp on five public datasets to demonstrate the effectiveness of our proposed method. The experiments show that our proposed method can achieve high accuracy with only a small portion of the training set. For example, it can achieve an average accuracy of 0.956 when using only 10% of the data training.

CCS CONCEPTS

• **Computer systems organization** → **Embedded systems**; *Redundancy*; Robotics; • **Networks** → Network reliability.

KEYWORDS

Log Analysis; Log Parsing; AI for Operations; Service Management;

ACM Reference Format:
Shimin Tao, Weibin Meng, Yimeng Chen, Yichen Zhu, Ying Liu, Chunning Du, and Tao Han, Yongpeng Zhao, Xiangguang Wang, Hao Yang. 2021. LogStamp: Automatic Online Log Parsing Based on Sequence Labelling. In *WAIN '21: 3rd International Workshop on AI in Networks and Distributed Systems, Nov 08–12, 2021, Politecnico Di Milano, Ital.* ACM, New York, NY, USA, 6 pages. https://doi.org/10.1145/1122445.1122456

1 INTRODUCTION

Logs are one of the most valuable data sources for large-scale services maintenance[1], which report service runtime status and help operators to find trace workflows. Logs have been widely applied for a variety of service management and diagnostic tasks. Prior research has proposed automated approaches to analyze logs, such as status monitoring [2], anomaly detection [3], failures prediction [4] and root cause analysis[5]. The fast-emerging AIOps (Artificial Intelligence for IT Operations) solutions also utilize operation logs as their input data[6].

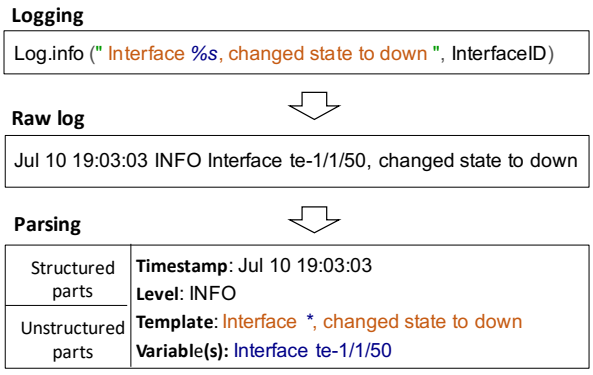

Figure 1: An illustrative example of log parsing from source code of logging to structure log

Logs are designed by developers and generated by logging statements (*e.g.*, *printf*(), *log.info*())) in the source code[7]. As shown in Fig.1, a logging function is composed of log level (*i.e.*, info), constant parts (*i.e.*, "Interface" and "change state to down"), and variables (*i.e.*, "InterfaceID"). Service and system generate raw logs by printing an unstructured text that contains constant text and specified variables (*e.g.*, "te-1/1/50"). Usually, the constant parts sketch out the event and summarize it, and variables vary from one log to another of the same template.

Since logs are often extensive in size (*e.g.*, Google and Facebook respectively generate 100 Petabyte and 10 Petabyte of log data per month[7]) and have free handwritten constructions[1], log analysis remains a significant challenge. To address the challenge of the large size of logs, researchers propose approaches for log compression[8]. However, most log compression approaches only aim to save storage space while not assisting when analyzing logs in practice. To address the challenge of log analysis, using rules (*e.g.*, source code[9] and regular expressions[10]) is a simple yet effective approach. However, the source code is not always available, and

*Weibin Meng (mengweibin3@huawei.com) is corresponding author

WAIN '21, Nov 08–12, 2021, Politecnico Di Milano, Italy

ACM ISBN 978-1-4503-XXXX-X/18/06…$15.00
https://doi.org/10.1145/1122445.1122456

designing regular expressions relies on domain knowledge, which cannot be used in practice. Therefore, automatic log parsing methods are getting attention. Researchers propose many approaches for automatically parsing raw logs into structured forms[11]. The main aim of log parsing is to find templates (constant parts) from logs and replace variables with variable placeholders. Recently, many data-driven log parsing approaches have been proposed. There are multiple techniques, such as clustering [12], longest common subsequence[13], frequent pattern mining[13, 14], heuristics parsing[15] and others[7]. However, log parsing still faces two challenges.

Firstly, operators continuously conduct software/firmware upgrades on services/systems to introduce new features, fix bugs, or improve performance[7], which can generate new types of logs. Most of the existing approaches do not support online analysis. A small number of approaches (*e.g.*, FT-tree[14], LogParse[7]) that support online parsing also have some shortcomings, or they cannot handle new types of words, or they need to be combined with other log parsing algorithms to complete. Therefore, *newly generated logs are difficult to process online* .

Besides, most existing log parsing approaches similar group logs and extracts templates for each group by keeping the same parts from logs and replacing different parts with placeholders. By default, log parsing is an unsupervised process. Parsers extract templates based on provided data instead of domain knowledge. Therefore, they only produce accurate results with sufficient historical log data. And, technically, the more data provided, the more accurate result they return. However, when a brand new service goes online, there are usually not enough historical logs to generate accurate templates. Therefore, *it's challenging to train a parsing model with small amounts of log data.*

To address the above challenges, we propose LogStamp. The key intuition is based on the following observations: When Operations reads the log, they mentally mark the words in the log to identify the template. In LogStamp, **we turn the log parsing problem into a sequence labelling problem** and find templates from logs online. LogStamp's contribution can be summarized as follows:

- LogStamp is an accurate online log parsing method. LogStamp can parse logs one by one, and has extremely high accuracy.
- LogStamp can train an accurate log parsing model based on a small amount of log data, which ensures that it can analyze online logs. Experiments show that it can achieve an average accuracy of 0.956 when using only 10% of the data training.

The rest of the paper is organized as follows: We discuss related works in Section 2 and propose our approach in Section 3. The evaluation is shown in Section 4. In Section 5, we discuss LogStamp's limitations and future works. Finally, we conclude our work in Section 6.

2 RELATED WORK

Logs play an important role in service management. Log parsing usually serves as the the first step towards automated log analysis [1].

The most straightforward approach is to use rules to parse logs, such as regular expressions. The rule-based log parsing methods rely on handcrafted rules provided by domain knowledge. Though straightforward, this kind of method requires a deep understanding of the logs, and a lot of manual efforts are needed to write different rules for different kinds of logs, which is not general. Commercial log analytic platforms (*e.g.*, Splunk, ELK, Logentries) also allow operators to efficiently manage and analyze large-scale logs by predefine rules. But they are only applicable to certain types of logs and are not universal.

Utilizing source code can parse logs accurate. For example, [9] employs source code to extract log templates for system problem detection. However, the source code is not always available, especially for commercial services.

To achieve the goal of automated log parsing, many data-driven approaches have been proposed. There are many categories of log parsing [1]. The first category is cluster-based approaches, which log template forms a natural pattern of a group of log messages. From this view, log parsing can be modeled as a clustering problem, such as LogSig [16]. Next is longest common subsequence. For example, Spell [13] uses the longest common subsequence algorithm to parse logs in a stream. Iterative partitioning is used in IPLoM [15]. Some methods use heuristics to extract templates. As opposed to general text data, log messages have some unique characteristics. Consequently, Drain [17] propose heuristics-based log parsing methods. The next category is frequent item mining, which is straightforward. Tokens, which regularly appear together in different log entries, are built into frequent itemsets. The parser obtains templates by looking up those itemsets. Log templates can be seen as a set of constant tokens that frequently occur in logs, such as FT-tree [14]. The final category is combined approaches. LogParse [7] combines existing unsupervised log parsing approaches and supervised machine learning approaches to generate templates for online logs. The idea of LogParse is similar to our paper. However, it's a pipeline workflow, which will be affected by the accuracy of traditional log parsing approaches because most log parsing approaches cannot achieve high accuracy based on a small number of logs.

3 DESIGN

In this section, we introduce the overall framework of our proposed LogStamp. The overview of the framework is shown in Fig. 2. We first present the offline part in our workflow in Section 3.1, then we will describe the online part in detail in Section 3.2.

3.1 Offline workflow

Given a set of historical logs, our goal is to build a tagger to identify if the incoming log is a template or variable. Previous works [7] use the template extraction method to obtain the templates from the logs. Then, a word classifier (i.e., SVM classifier) is adopted to label each word in the logs. There are two drawbacks to such methods. First, the accuracy of labeling depends on the quality of the extracted template. If the template extraction method fails to separate the templates from the raw logs, the pseudo label assign by the classifier would be meaningless and cause failure on log parsing. Secondly, prior works only utilize templates to train the word classifier. Generally, log data contains critical information on both word-level and sentence-level (i.e., sentence order). For example, in log anomaly detection task, a common way to detect

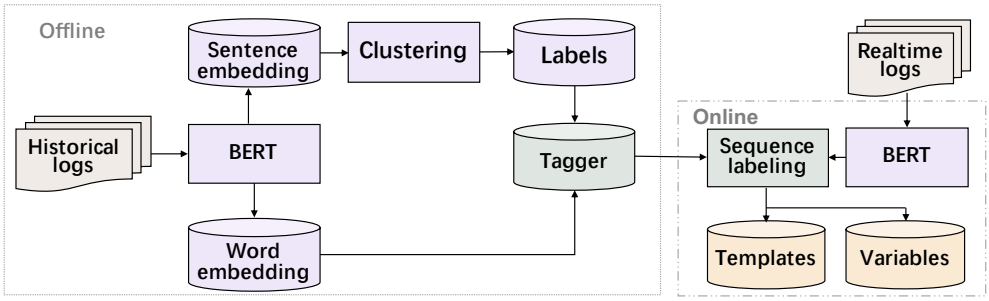

Figure 2: The workflow of LogStamp

the anomalous logs in the log data is to see if the log orders are correct. If we have received a log saying that "*Vlan-interface, change state to up*" and no message of "*Vlan-interface, change state to down*" is followed in a certain time period, we will recognize such log as anomalous log. And because the word-level embedding only focus on the single word, it fails to effective parse such logs .Therefore, learning logs feature from the sentence level is important.

In this paper, we introduce a coarse to the fine framework to generate accurate pseudo labels. First of all, a pretrained bidirectional transformer is adopted to extract the feature representation of log data. Because the structure of raw logs is different from the natural language, we need to fine-tune the BERT [18] using our data. Note that finetuning the BERT does not need any label. Then we use a dual-path framework to get both coarse level embedding and fine level embedding. On the coarse level, we expect the sentence embedding can reflect the nature of different logs. For instance, the above example of two logs have similar structure, and most of the words in these two logs are the same. However, the meaning of these two logs are completely different. The coarse level feature learns the inherent relations between the words, thus output two embeddings with distant similarity. The sentence embedding can be further grouped into a number of clusters.

In general, one can exploit any clustering algorithm that can split the sentence into clusters according to their embedding features. Our approach is to use DBSCAN [19]. After we get the clusters, we count the frequency of word appearance in each clustering. We mark it as template if the number of appearance is larger than the threshold, variables otherwise. As such, we obtain the labels for each word.

For the fine-grained level, we used the fine-tuned BERT to output word embeddings. For each word embedding, we have its corresponding label from the step above. Given a set of word embeddings and word labels, we can train a classifier that serves as a tagger. As we trained via a deep neural network, this tagger can accurately parse the logs without the interference introduced by the wrong pseudo labels.

3.2 Online workflow

In real-time systems, systems may generate new log templates online; therefore, building a robust online workflow is critical for real scenario deployment. Our online workflow is simple. Given real-time logs, which can be either a piece of logging information or a set of new logs, we reuse the BERT model to extract the word embedding from the new logs. Then the tagger that is trained in the offline stage will predict a label for the logs. As a result, we can immediately know whether the specific words are templates or variables. We will show that our online framework is simple yet surprisingly effective under most of the circumstance.

Table 1: Detail of the datasets

Datasets	Description
HDFS	Hadoop distributed file system
Proxifier	Proxifier software
ZooKeeper	ZooKeeper service
BGL	Blue Gene/L supercomputer
Hadoop	Hadoop MapReduce job

4 EVALUATION

In this section, we evaluate our approach using public log datasets and aim to answer the following research questions:

- RQ1: How effective is LogStamp in log parsing?
- RQ2: Can LogStamp achieve accurate results based on a small amount of log data?
- RQ3: How much can the BERT and tagger contribute to the overall performance?

4.1 Experiment Setting

In this section, we evaluate the performance of LogStamp. The datasets, baselines, evaluation metrics and experimental setup of the experiments are as follows.

4.1.1 Datasets. We conduct experiments over five public log datasets from distributed systems, which are BGL [1], HDFS [20], ZooKeeper [21], Proxifier [1] and Hadoop [12]. The detailed information of these datasets is listed in Table 1. For each dataset, [1] sampled logs and manually labelled each log's template, which serves as the ground truth for our evaluation.

4.1.2 Baselines. To demonstrate the performance of LogStamp, we have applied/implemented seven template extraction methods: FT-tree [14], Drain [17], Spell [13], LogSig [16], LogParse [7], MoLFI [22] and IPLoM [15] . The parameters of these methods are all set best for accuracy. LogParse [7] can incorporate any existing

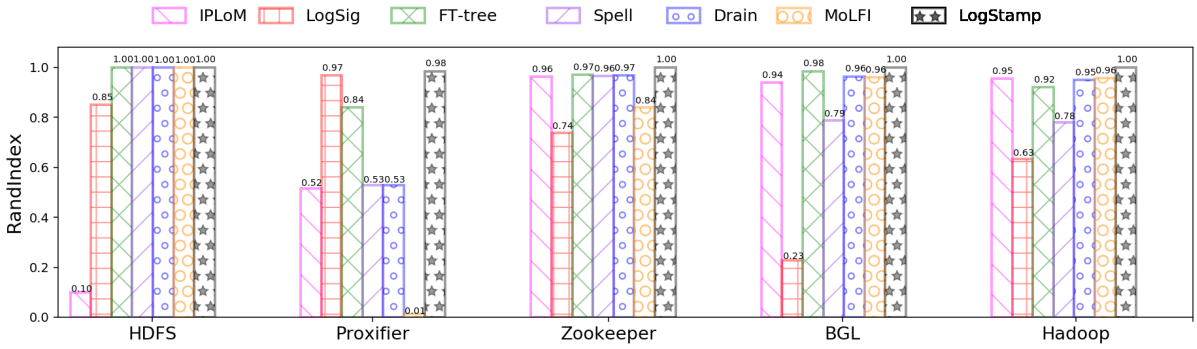

Figure 3: Comparison of the accuracy of offline log parsing between LogStamp and six baselines when they are trained by all offline logs

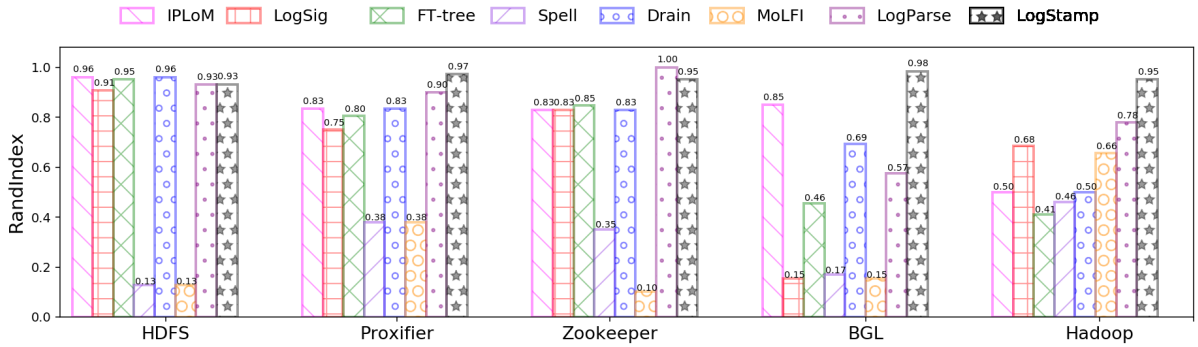

Figure 4: Comparison of the accuracy of online log parsing between LogStamp and seven baselines when they are trained by 10% offline logs

template extraction method, our paper utilized Spell to initialize LogParse.

For BERT, we use three versions of BERT, *i.e.*, BERT-base, BERT-tiny and BERT-small. For tagger in LogStamp, we compare the performances of GCN, CNN, LSTM and RNN.

4.1.3 Evaluation Metrics. We apply *RandIndex* [23] to quantitatively evaluate the accuracy of template extraction. *RandIndex* is a popular method for evaluating the similarity between two data clustering techniques or multi-class classifications. What's more, *RandIndex* is applied to evaluating existing template extraction methods in the literature, such as in [7].

For each template extraction method, we evaluate its accuracy by calculating the *RandIndex* between the manual classification results and the templates learned by it. Specifically, among the template learning results of a specific method, we randomly select two logs, *i.e.*, x and y, and define TP, TN, FP, FN as follows. TP: x and y are manually classified into the same cluster and they have the same template; TN: x and y are manually classified into different clusters and they have different templates; FP: x and y are manually classified into different clusters and they have the same template; FN: x and y are manually classified into the same cluster and they have different templates. Then *RandIndex* can be calculated using the above terms as follows: $RandIndex = \frac{TP+TN}{TP+TN+FP+FN}$.

4.1.4 Experimental Setup. We conduct experiments on a Linux server with Intel Xeon 2.40 GHz CPU and 64G memory.

4.2 Evaluation Results

1) RQ1: How effective is LogStamp in log parsing?

We first compare accuracies of existing log parsing methods[1] and LogStamp when we extract templates from historical logs. The comparison results are shown in Fig. 3. We find that most log parsing methods are highly accurate in extracting templates from historical logs. However, the accuracy of existing parsers is not always consistent. In other words, the selection of log data impacts the parsing accuracy [1]. Parsers may have a good evaluation result with up to 90% of accuracy and an unacceptable bad outcome down to 50% depending on different input datasets (*e.g.*, Proxifier). Meanwhile, we find LogStamp still achieves high accuracy (the average accuracy is more than 0.999) on different datasets. Therefore, we can directly use the label results to train a tagger.

To demonstrate the performance of LogStamp in supporting online parsing and simulate the launch of new services, for each dataset, we apply each log parsing method to extract templates from 10% of their logs. Fig. 4 shows the comparative results. LogStamp

[1]LogParse's offline step utilized other log parsing methods. It doesn't have its own offline parsing.

Table 2: Offline accuracy of LogStamp with different BERT versions

Methods	Datasets				
	HDFS	Proxifier	Zookeeper	BGL	Hadoop
BERT-tiny	0.9999	0.9356	0.9998	0.9950	0.9988
BERT-base	0.9999	0.9836	0.9998	0.9994	0.9987
BERT-small	0.9999	0.9840	0.9998	0.9979	0.9988

Table 3: Online accuracy of LogStamp with different BERT versions

Methods	Datasets				
	HDFS	Proxifier	Zookeeper	BGL	Hadoop
BERT-tiny	0.8888	0.9042	0.9906	0.9788	0.9762
BERT-base	0.8798	0.9141	0.9760	0.9816	0.9637
BERT-small	0.9147	0.8820	0.9851	0.9586	0.9752

achieves the best performance. Specifically, the accuracy of LogStamp on each dataset is 0.956.

2) RQ2: Can LogStamp achieve accurate results based on a small amount of log data?

As shown in [1], the accuracy of existing parsers is not always consistent, both for the datasets and the percentage of training data. To demonstrate how stable LogStamp is to the scale of training data, Fig. 5 shows the log parsing accuracy of LogStamp on the five datasets, as the percentage of training data increases from 10% to 90%, respectively. The results show that LogStamp is stable to different scales of training logs and can achieve high log parsing accuracy when trained based on a small scale of training data.

3) RQ3: How much can the BERT and tagger contribute to the overall performance?

LogStamp incorporates two modules: BERT and taggers. In this RQ, we evaluate the effectiveness of different version of each module. Firstly, we compare LogStamp with BERT-base, BERT-small and BERT-tiny. Table 2 and Table 3 show the performance of LogStamp in the offline stage and the online stage, respectively. We find that three versions of BERT achieve similar performance, which means that LogStamp doesn't need to spend time adjusting the effect of BERT. Then, in Table 4, we compare LogStamp with different taggers, *i.e.*, GCN, RNN, LSTM and CNN. We find that LSTM achieves the best performance on all datasets. Because LSTM is more suitable to natural language processing, and sequence labelling is a problem in natural language processing.

5 DISCUSSION AND FUTURE WORK

Thanks to BERT for its powerful ability to capture both sentence embedding of log sentences for clustering and word embedding for distinguishing between templates and variables in log. However, during experiments, it is observed that syntactic structure and semantic information contained in log sentences often vary considerably compared to those sentences used to train BERT. One deduction is that if the BERT model is fine-tuned on log datasets with masked language modeling, it might better understand log

Table 4: Online Accuracy of LogStamp with different taggers

Methods	Datasets				
	HDFS	Proxifier	Zookeeper	BGL	Hadoop
GCN	0.8888	0.9042	0.9906	0.9788	0.9762
RNN	0.9822	0.9180	0.9790	0.9978	0.9962
LSTM	0.9949	0.9998	0.9998	0.9996	0.9974
CNN	0.9921	0.9164	0.9998	0.9996	0.9974

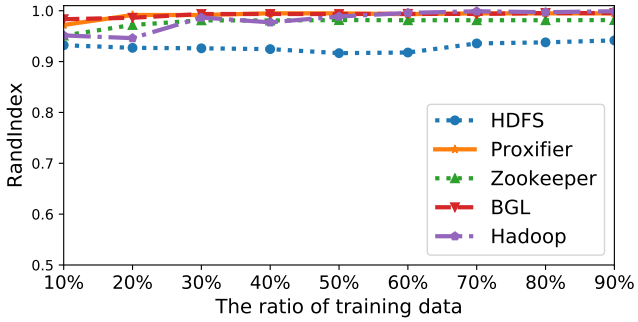

Figure 5: The log parsing accuracy of LogStamp as the ratio of training data changes

sentences and thus have higher accuracy in offline and online log parsing. Yet, the experiment result does not prove the deduction to be correct. By fine-tuning the BERT model with log sentences of each system for 1-3 epochs, the online clustering rand index does not seem to be steadily improved.

We will continue to study how to apply better pre-trained language models to log template extractions in our future work. More abundant logs will be used to fine-tune BERT or to train a BERT from the beginning instead of directly loading weights of model pre-trained with dissimilar vocabularies, *e.g.*, from Wikipedia or books. Besides, as log sentences usually have a more unified structure, we will also attempt to design a more concise model structure based on BERT to achieve higher efficiency in online log parsing to deal with higher concurrency.

6 CONCLUSION

In this paper, we propose LogStamp, an online log parsing approach. Different from the prior log parsing approach, LogStamp takes semantic into consideration and turns the log parsing problem into a sequence labelling problem. LogStamp supports a training model based on a small number of historical logs. Experimental results on public log datasets have validated the accuracy and stability of LogStamp.

7 ACKNOWLEDGMENT

The work was supported by National Key R&D Program of China (Grant No. 2018YFB1800405). We thank the anonymous reviewers for their valuable feedback. We thank Yi Zhou, Yuming Xie and Ying Qin for their great help.

REFERENCES

[1] Jieming Zhu, Shilin He, Jinyang Liu, Pinjia He, Qi Xie, Zibin Zheng, and Michael R Lyu. Tools and benchmarks for automated log parsing. In *Proceedings of the 41st International Conference on Software Engineering(ICSE)*, pages 121–130, 2019.

[2] Subhendu Khatuya, Niloy Ganguly, Jayanta Basak, Madhumita Bharde, and Bivas Mitra. Adele: Anomaly detection from event log empiricism. *IEEE Conference on Computer Communications (INFOCOM)*, 2018.

[3] Weibin Meng, Ying Liu, Yichen Zhu, Shenglin Zhang, Dan Pei, et al. Loganomaly: Unsupervised detection of sequential and quantitative anomalies in unstructured logs. In *Proceedings of the Twenty-Eighth International Joint Conference on Artificial Intelligence, IJCAI-19.*, volume 7, pages 4739–4745, 2019.

[4] Shenglin Zhang, Ying Liu, Weibin Meng, Zhiling Luo, Jiahao Bu, Sen Yang, Peixian Liang, Dan Pei, Jun Xu, Yuzhi Zhang, et al. Prefix: Switch failure prediction in datacenter networks. *Proceedings of the ACM on Measurement and Analysis of Computing Systems(SIGMETRICS)*, 2(1):2, 2018.

[5] Satoru Kobayashi, Kensuke Fukuda, and Hiroshi Esaki. Mining causes of network events in log data with causal inference. In *2017 IFIP/IEEE Symposium on Integrated Network and Service Management (IM)*, pages 45–53. IEEE, 2017.

[6] Hetong Dai, Heng Li, Che Shao Chen, Weiyi Shang, and Tse-Hsun Chen. Logram: Efficient log parsing using n-gram dictionaries. *IEEE Transactions on Software Engineering*, 2020.

[7] Weibin Meng, Ying Liu, Federico Zaiter, Shenglin Zhang, Yihao Chen, Yuzhe Zhang, Yichen Zhu, En Wang, Ruizhi Zhang, Shimin Tao, et al. Logparse: Making log parsing adaptive through word classification. In *2020 29th International Conference on Computer Communications and Networks (ICCCN)*, pages 1–9. IEEE, 2020.

[8] Jinyang Liu, Jieming Zhu, Shilin He, Pinjia He, Zibin Zheng, and Michael R Lyu. Logzip: Extracting hidden structures via iterative clustering for log compression. In *2019 34th IEEE/ACM International Conference on Automated Software Engineering (ASE)*, pages 863–873. IEEE, 2019.

[9] Wei Xu, Ling Huang, Armando Fox, David Patterson, and Michael I. Jordan. Detecting large-scale system problems by mining console logs. In *ACM Sigops Symposium on Operating Systems Principles*, pages 117–132, 2009.

[10] Weibin Meng, Ying Liu, Shenglin Zhang, Federico Zaiter, Yuzhe Zhang, Yuheng Huang, Zhaoyang Yu, Yuzhi Zhang, Lei Song, Ming Zhang, et al. Logclass: Anomalous log identification and classification with partial labels. *IEEE Transactions on Network and Service Management*, 2021.

[11] Steven Locke, Heng Li, Tse-Hsun Peter Chen, Weiyi Shang, and Wei Liu. Logassist: Assisting log analysis through log summarization. *IEEE Transactions on Software Engineering*, 2021.

[12] Qingwei Lin, Hongyu Zhang, Jian-Guang Lou, Yu Zhang, and Xuewei Chen. Log clustering based problem identification for online service systems. In *Proceedings of the 38th International Conference on Software Engineering Companion (ICSE)*, pages 102–111. ACM, 2016.

[13] Min Du and Feifei Li. Spell: Streaming parsing of system event logs. In *2016 IEEE 16th International Conference on Data Mining (ICDM)*, pages 859–864. IEEE, 2016.

[14] Shenglin Zhang, Weibin Meng, Jiahao Bu, Sen Yang, Ying Liu, Dan Pei, Jun Xu, Yu Chen, Hui Dong, Xianping Qu, et al. Syslog processing for switch failure diagnosis and prediction in datacenter networks. In *2017 IEEE/ACM 25th International Symposium on Quality of Service (IWQoS)*, pages 1–10. IEEE, 2017.

[15] Adetokunbo AO Makanju, A Nur Zincir-Heywood, and Evangelos E Milios. Clustering event logs using iterative partitioning. In *Proceedings of the 15th ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 1255–1264. ACM, 2009.

[16] Liang Tang, Tao Li, and Chang-Shing Perng. Logsig: Generating system events from raw textual logs. In *Proceedings of the 20th ACM international conference on Information and knowledge management*, pages 785–794. ACM, 2011.

[17] Pinjia He, Jieming Zhu, Zibin Zheng, and Michael R Lyu. Drain: An online log parsing approach with fixed depth tree. In *2017 IEEE International Conference on Web Services (ICWS)*, pages 33–40. IEEE, 2017.

[18] 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.

[19] Martin Ester, Hans-Peter Kriegel, Jörg Sander, Xiaowei Xu, et al. A density-based algorithm for discovering clusters in large spatial databases with noise. In *Kdd*, volume 96, pages 226–231, 1996.

[20] Wei Xu, Ling Huang, Armando Fox, David Patterson, and Michael Jordan. Largescale system problem detection by mining console logs. *Proceedings of SOSP'09*, 2009.

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

[22] Salma Messaoudi et al. A search-based approach for accurate identification of log message formats. In *Proceedings of the 26th Conference on Program Comprehension*, pages 167–177. ACM, 2018.

[23] William M Rand. Objective criteria for the evaluation of clustering methods. *Journal of the American Statistical association*, 66(336):846–850, 1971.