LogNG: an Online Log Parsing Method Based on N-Gram

Xiangrui Liu, Shi Ying, Xinquan Ge, Shengkang Hu and Tiangang Li

February 8, 2022

LogNG: An Online Log Parsing Method Based on N-gram

1st Xiangrui Liu
School of Computer Science
Wuhan University
Wuhan, China
2020202110102@whu.edu.cn

2nd Shi Ying
School of Computer Science
Wuhan University
Wuhan, China
yingshi@whu.edu.cn

3rd Xinquan Ge
School of Computer Science
Wuhan University
Wuhan, China
2020202110054@whu.edu.cn

4th Shengkang Hu
School of Computer Science
Wuhan University
Wuhan, China
2019202110082@whu.edu.cn

5th Tiangang Li
School of Computer Science
Wuhan University
City, Country
tiangangli@whu.edu.cn

***Abstract*—The first step in automatic log analysis is log parsing. The number of logs exploded with the increase in system size. Manual analysis of logs has become a difficult problem. To solve this problem, we proposed logNG, an online log parsing method based on N-gram that can efficiently parse logs in a streaming manner without the requirement for historical data training. When log messages are input in a stream, we first divide log messages of different lengths into different log groups. We'll use an intuitive and simple assumption: If continuous multiple different tokens appear between log messages, these log messages belong to different log template. For each log group, we will use N-gram for template matching to further group log messages within our assumption. We evaluate and compare logNG with other log parsers on public data sets. The results of the experiments reveal that logNG can achieve the highest accuracy and efficiency.**

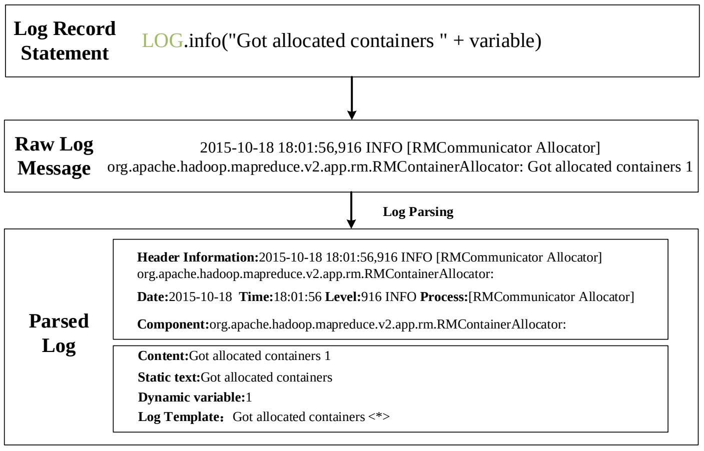
***Index Terms*—Log parsing, Online algorithm, N-gram**

I. Introduction

Logs play an important role in modern software systems, but mining its value is still a huge challenge [1], [2], [3], [4]. Log parsing is the first step in automatic log analysis. The quality of log parsing greatly affects the downstream tasks of automatic log analysis [5], [6], [7]. The definition of log parsing is to convert unstructured data into structured data [8], [9]. Its purpose is to separate header information (including date, time, level, etc.) and content (including static text and dynamic variables) [10], [11], [12].

As shown in "Fig. 1", when the system is running, a log record statement will generate a raw log message [13], which usually consists of header information and content. The header information is usually composed of date ("*2015-10-18*"), time ("*18:01:56*"), level ("*916 INFO*") and other parts.

Log parsing requires more content ("*Got allocated containers*") than the header information which can be filtered out by regular expressions. The content consists of the static text of the log record statement (""*Got allocated containers*") and its designated dynamic variable ("*1*"). The log template corresponding to this log message is (""*Got allocated containers <*>*") The position of the dynamic variable in the log template is marked with a wildcard "<*>".



```
LOG.info("Got allocated containers " + variable)
```

```
2015-10-18 18:01:56,916 INFO [RMCommunicator Allocator]
org.apache.hadoop.mapreduce.v2.app.rm.RMContainerAllocator: Got allocated containers 1
```

Header Information:2015-10-18 18:01:56,916 INFO [RMCommunicator Allocator] org.apache.hadoop.mapreduce.v2.app.rm.RMContainerAllocator:
Date:2015-10-18 **Time:**18:01:56 **Level:**916 INFO **Process:**[RMCommunicator Allocator]
Component:org.apache.hadoop.mapreduce.v2.app.rm.RMContainerAllocator:

Content: Got allocated containers 1
Static text: Got allocated containers
Dynamic variable:1
Log Template: Got allocated containers <*>

Fig. 1. The Log Parsing Process of a Raw Log Message from Hadoop.

In this paper, we propose an online automatic log parsing method logNG, which accurately and efficiently parses the raw log messages in a streaming manner. logNG automatically extracts log templates from raw log messages without source code and historical log data.

We evaluated logNG and other log parsers on real log data sets collected by the LogPai team[1] [14]. logNG achieved the highest results on most of the data sets, and it was also very fast in running time.

In general, our work mainly has the following contributions:

- This paper proposes logNG, an online automatic log parsing method. The log messages we input in the form of a stream are divided into different log groups according to their length. For each log group, we use N-gram to further divide log messages for template matching.

[1]https://github.com/logpai/logparser

- Our method not only solves the problem of manually parsing log templates, but also is an online method that does not require collection of historical data for training.
- The experimental results on real log data sets prove the accuracy and high efficiency of logNG.

II. Related work

Rule-based log parsing relies on artificial heuristic rules (basically in the form of regular expressions) to parse logs. However, this approach is not feasible due to rapid development of log size [15], [16], [17], [18], [19].

Log parings based on source code has been supported by some research [20], [21]. However, this method is actually difficult to achieve because of unavailability of source code.

Log parsing based on data mining does not require source code, but uses various data mining techniques to separate dynamic variables and static text by mining the characteristics in the log [22].

LKE [23] is a representative algorithm for log parsing. In this offline parser, log messages are hierarchically clustered using weighted edit distance, and log keys are generated from the generated cluster. The log key corresponds to the log print statement. After the log message is converted to the log key, a finite state automaton is learned from the training log sequence.

LogMine [24] is an unsupervised framework. It only scans log messages once and works in an iterative manner to generate a pattern hierarchy that can be extracted from a set of log messages quickly and efficiently High-quality mode, which can process millions of log messages in a few seconds.

MoLFI [25] is a tool for solving the problem of log message format identification. It reconstructs the problem of log message identification into a multi-objective problem and uses an evolutionary method to solve this problem.

SHISO [26] is an online method of mining log formats and retrieving log types and parameters. It creates a structured tree by using nodes generated from log messages.

III. METHODOLOGY

A. Method Overview

1) Data Structure: A good data structure can be more convenient for us to analyze. We introduced a data structure, the log group, as shown in "Fig. 2".

The log group is a data structure with four attributes, including *Template*, *Length*, *TemplateID*, and *LogIDList*. In this paper, italicized and capitalized words indicate attributes.

Before the logs are entered in the form of a stream, we first create an empty log group list. When log messages are continuously input, logNG will create log groups and add them to the log group list.

Each attribute of the log group has its role. As the name implies, *Template* and *Length* are the log template parsed by our parser and the number of tokens. Taking the log group in "Fig. 2" as an example, *Length* is 4 and *Template* is "*Verification succeeded for* <*>".

When the log group is created for the first time, we directly assign the content of the log message to *Template*. At this time, logNG has not distinguished between static text and dynamic variables. When the next log message comes, we will match it with *Template* of the existing log group. If the match is successful, logNG will compare *Template* with this log message. logNG will recognize static text and dynamic variables, and finally update.

TemplateID refers to the ID of *Template*. The log group list is empty at the beginning. There is no log group, and *TemplateID* value does not exist at this time. When a new log group is generated, logNG assign *TemplateID* to 1. After that, whenever a new log group appears, logNG will assign a new *TemplateID*, the value of which is the previous *TemplateID* plus 1, so that *TemplateID* value is equivalent to the sequence number of this log group in the log group list. For example, *TemplateID* in "Fig. 2" is 13, indicating that this is the 13th log group and log template generated and identified by logNG.

LogIDList is a list of the log ID. Each log message has a corresponding log ID. When the log message matches the log group, the *LogIDList* of the log group will add the ID value of the log message. For example, *LogIDList* in "Fig. 2" is [1,22,56,168,245. ..], this means that log messages with log ID 1, 22, 56, 168, 245, etc. match *Template* with *TemplateID* 13.

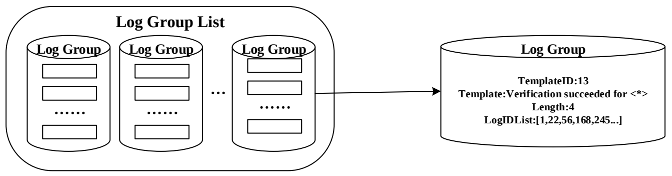


Fig. 2. Data Structure.

2) Hierarchical Structure: logNG is divided into 4 hierarchical structures from beginning to end, as shown in "Fig. 3".

When the raw log message is input into the logNG, it first passes through the preprocessing layer. The preprocessing layer will filter out the header information part of the raw log message first, and mark some fixed format data (such as IP address, block ID, etc.) as "<*>", to facilitate the subsequent parsing process.

In the length layer, logNG will divide log messages into different log groups by length. Taking "Fig. 3" as an example, log messages can be divided into groups of the length less than 3, the length less than 4, and the length equal to 5 and so on.

It should be noted that when the length of the log message is less than N, logNG cannot parse this kind of log message because the length is not enough. When logNG encounters this situation, it first checks whether the log group corresponding to this log message has been saved in the log group list. If so, it only adds the ID of the log message to *LogIDList* of the log group. If not, it creates a new log group directly.

In the matching layer, logNG will make N-gram judgments on log messages and divide them into more detailed log groups. This is because different log templates may exist in log groups of the same length.

In the update layer, logNG identifies the static text and dynamic variable parts in *Template* of the log group, and updates it.

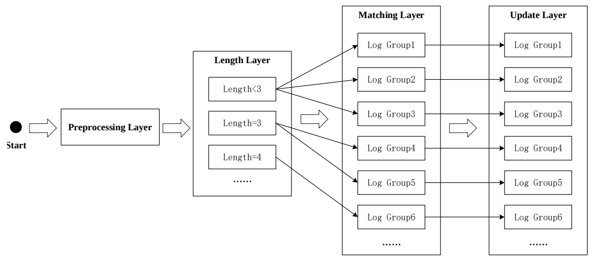


Fig. 3. Hierarchical Structure.

B. Step 1: Preprocessing

logNG starts with an raw log message every time. Preprocessing is the first step of our method. Previous studies [13] have shown that preprocessing can improve the accuracy of log parsing. Preprocessing can filter out the header information of the raw log message and extract the content. As mentioned above, the header information of the raw log message is usually composed of date, timestamp, level and so on. Because most logs of the same software system follow the general format configured by the log library, we can easily delete the header information and obtain the content directly. In addition, some fixed-format data can be marked as "<*>" using predefined regular expressions, such as IP address or block ID.

C. Step 2: Length Judgment

When a raw log message is preprocessed, this method will divide it according to the length. A research [27] has shown that extracting log templates from log messages of the same length can easily achieve good results. If this raw log message is the first data in the log data set, then the log group list is empty. logNG directly creates a new log group and adds it to the log group list. If the log group list is not empty, compare the length of this log message with the length of all log groups. If the match is successful, add it to the log group list, otherwise create a new log group.

Take "Fig. 4" as an example. The log message is "*Got allocated containers 1*", and *TemplateID* is set to 1, and *LogIDList* is *[1]*, and *Length* is 4, and *Template* is log message itself, that is, "*Got allocated containers 1*". If the log group list is not empty, compare the length of the log message with *Length* of all log groups. If there is no log group with the same length, then directly create a new log group and add it to the log group list. If there are log groups of the same length, proceed to the next step of judgment.

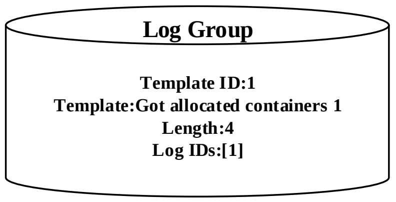


Fig. 4. The First Log Group in the Log Group List.

D. Step 3: N-gram Matching and Update

Length layer can filter out most of the log messages that do not belong to the same log template. However, there is still a problem with this division, because some log messages have the same length, but belong to different log templates, which shows that we need more detailed matching and identification methods.

In this section, we will use an intuitive and simple assumption: If continuous multiple different tokens appear between log messages, these log messages belong to different log template. N-gram [28] must be used under this assumption. When logNG sets N to 3, we believe that when different tokens appear three times in a row between two log messages, the two log messages do not belong to the same log template. The setting of N is defined by the user. For the convenience of discussion, logNG sets N to 3.

We first obtain the 3-gram lists of the log message and *Template* in the log group respectively. For the two 3-gram lists, we compare each pair of 3-grams one by one in order of position. We check whether each pair of 3-grams is completely different. When no pair of gram (or token) in each pair of 3-grams is the same, we think they are completely different. Note that "<*>" is a variable, and they are different from each other by default. When two 3-gram lists have at least one pair of 3-grams that are completely different, we think that this log message does not belong to this log group, so we skip this log group and compare it with the next log group with the same length. Otherwise we think this log message belongs to this log group.

"Fig. 5" shows the comparison of the two cases respectively. We bold the different parts of each pair of 3-grams. In the first case, *Template* is "*PacketResponder 1 for block <*> terminating*" and the log message is "*PacketResponder 2 for block <*> terminating*". In the pair of 3-gram lists obtained by them, there are no completely different pair of 3-grams. For example, in the first pair 3-grams, although token "*1*" and token "*2*" are different, but the other parts are the same. So the first pair 3-grams are not completely different. We have reason to believe this log message belongs to this log group.

In the second case, *Template* is still "*PacketResponder 1 for block <*> terminating*" and the log message is "*1 failures on node MININT-FNANLI5.fareast.corp.microsoft.com*". In the pair of 3-gram lists obtained by them, the first pair of 3-grams (i.e., "*PacketResponder 1 for*" and "*1 failures on*") are completely different, which means that this log message does not belong to this log group and it is no longer necessary to continue compare.

When we determine that a log message belongs to a certain log group, in addition to adding the ID of this log message to the *LogIDList*, we also need to update the *Template*. To update the log group *Template* is to mark the tokens that are different between the log message and the log group *Template* as “<*>”.

Taking the first case of “Fig. 5” as an example, only the second token(i.e., “*1*” and “*2*”) is different between the log message and *Template* of the log group, so we only need to replace the token in this place with “<*>”. Finally, we get the updated log *Template*: ”*PacketResponder “<*> for block <*> terminating*”.

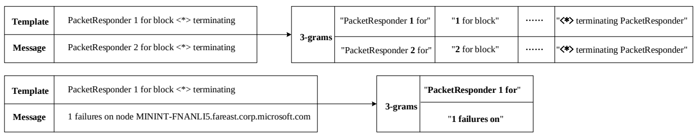

Template
PacketResponder 1 for block <*> terminating
Message
PacketResponder 2 for block <*> terminating
3-grams
"PacketResponder 1 for"
"1 for block"
······
"<*> terminating PacketResponder"
"PacketResponder 2 for"
"2 for block"
······
"<*> terminating PacketResponder"
Template
PacketResponder 1 for block <*> terminating
Message
1 failures on node MININT-FNANLI5.fareast.corp.microsoft.com
3-grams
"PacketResponder 1 for"
"1 failures on"

Fig. 5. N-gram Matching.

IV. Evaluation

A. Experimental Setup

1) Comparison: In order to illustrate the effectiveness of our method, we compare the accuracy and efficiency of logNG with the existing four log parsing methods, including three offline log parsers and one online log parser. The brief information of these log parsing methods is shown in “Table. I”, and is introduced in Section II.

TABLE I
Five Comparison Methods for Experiments

Log Parser	Pattern	Method
LKE	Offline	Clusterinig
LogMine	Offline	Clusterinig
MoLFI	Offline	Evolutionary Algorithms
SHISO	Online	Clusterinig

2) Log Data Set: The LogPai team [14] provides convenience to researchers in the industry and academia. They store large log data sets from different systems on their Loghub[2] [29]. These systems include distributed systems, supercomputers, mobile systems, independent software and server applications. Loghub collected a total of 440 million log messages with a size of 77GB, which is now the largest collection of log data sets [14]. We will conduct experiments on 10 public data sets provided by the LogPai team. For each data set, the LogPai team randomly selected 2000 log messages and manually marked the log template corresponding to each log message as the groud-truth for our evaluation. The information of these data sets is listed in “Table. II”. The Events(2k) table shows the number of events in 2000 log messages.

[2]https://github.com/logpai/loghub

TABLE II
Proportion of Continuous Dynamic Variables

Dataset	Description	Events(2k)
HDFS	Distributed System	14
Hadoop	Distributed System	114
Spark	Distributed System	36
Zookeeper	Distributed System	50
BGL	Supercomputer	120
HPC	Supercomputer	46
Thunderbird	Supercomputer	149
HealthApp	Mobile System	75
Apache	Server Application	6
Proxifier	Standalone Software	8

TABLE III
Parsing Accuracy of Log Parsing Methods

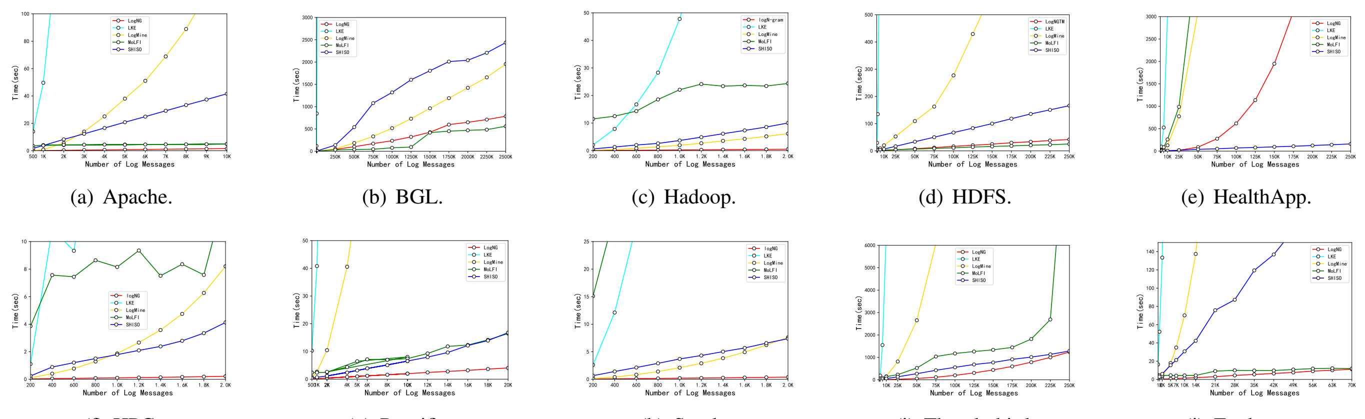
	logNG	LKE	LogMine	MoLFI	SHISO
HDFS	**1**	**1**	0.8505	0.9975	0.9975
Hadoop	**0.9545**	0.6695	0.8695	0.8535	0.867
Spark	**0.92**	0.6335	0.5755	0.418	0.906
Zookeeper	**0.987**	0.4375	0.6875	0.839	0.66
BGL	**0.9545**	0.6455	0.7245	0.9385	0.711
HPC	**0.904**	0.574	0.784	0.8245	0.3245
Thunderbird	**0.9445**	0.8125	0.9185	0.648	0.576
HealthApp	**0.9915**	0.5915	0.6865	0.535	0.397
Apache	**1**	**1**	**1**	**1**	**1**
Proxifier	**1**	0.495	0.5165	0.013	0.5165
Average	**0.9509**	0.6859	0.7613	0.7067	0.6956

3) Evaluation Metric: We will prove the effectiveness from the perspectives of parsing accuracy (PA). F-measure is a weighted harmonic average of precision and recall. It is a commonly used evaluation standard and is often used to evaluate the quality of classification models. But PA is a more stringent metric than F-measure. [13] defined PA as the ratio of the number of log messages correctly parsed to the total number of log messages. If and only if the log *Templates* generated is exactly the same as ground truth, we think it is parsed correctly. Even if there is only one log message that does not match correctly, we still think that the parsing of the log template has failed.

4) Experimental Environment: All our experiments were performed on a Linux server running 64-bit Centos 7.4, equipped with 16-core Intel(R) Xeon(R) Silver 4216 CPU @ 2.10GHz and 32GB DDR4 2666. We support the implementation of logNG in Python 3.8.6. We run each experiment 10 times to get the average value to avoid bias.

B. Result Analysis

1) Effectiveness: Accuracy shows the ability of the log parser to correctly match the raw log message with the log template. A study [30] shows that accurate parsing of log messages into templates is essential for many log processing methods, and parsing errors are likely to cause the performance of subsequent downstream tasks to degrade. We compare logNG with 4 log parsing methods, including 3 offline methods and 1 online methods. Generally speaking, the offline method will be more accurate than the online method, because the



(a) Apache. (b) BGL. (c) Hadoop. (d) HDFS. (e) HealthApp.

(f) HPC. (g) Proxifier. (h) Spark. (i) Thunderbird. (j) Zookeeper.

Fig. 6. Running Time of Log Parsing Methods on Data Sets in Different Size

offline method can use all the raw log messages from the beginning. But the online method is not, it can only gradually adjust itself [13].

“Table. III” show the comparison of experimental data. The bold data indicates the highest performance value that can be achieved on this data set.

From the results of PA, logNG can achieve the highest PA on 10 data sets, which are all above 0.9. The average PA of logNG reaches 0.9509, which is also the highest among all log parsers. The second-ranked parser is LogMine, with an average PA of 0.7613. There is an obvious gap between logNG and LogMine. logNG is not only significantly more effective than other log parsers, but also more comprehensive. It can achieve good or even the best results on each data set.

2) Efficiency: Efficiency is also a very important indicator. The current log size is huge and the production speed is fast, which puts forward requirements for the efficiency of the log parser. In order to illustrate the running speed of logNG, we will also conduct experiments on 10 real data sets of different sizes and compare them with the other four log parsing methods.

“Fig. 6” shows the running time of the five methods. The red line in the figure is the running time of logNG on log data sets of different sizes. The results show that logNG can achieve the fastest speed on most data sets. Even if it is not optimal for the three data sets of BGL, HDFS and HealthApp, it can still be ranked second. On the whole, logNG running time is maintained at a high level.

In addition, our method linearly increases with the increase of log size on most data sets. However, LKE and LogMine are often squared with the log size, causing their images to be out of range. Because their time complexity is $O(n^2)$. The time complexity makes them unable to complete the log parsing task in limited time. And online parsing methods (i.e., SHISO, logNG) process log messages one by one, so they enjoy linear time complexity $O(n)$ which makes them run faster. Although LogMine can achieve the second place in accuracy, it costs a lot of time.

V. CONCLUSION

In this work, we propose an online log parsing method based on N-gram. We use an intuitive assumptions: Log messages of the same log template will not have consecutively different tokens. With the assumption, we have realized the processing of raw log messages in a stream. The advantage of this method is that no historical data training is required. In order to prove the effectiveness and efficiency of logNG, we conducted experiments on 10 real-world log data sets collected by the LogPai team [14]. The final results show that logNG can achieve the best accuracy on most data sets, and the running speed is faster than other log parsing methods. However, during our research, we found that some dynamic variables of log templates have multiple value forms. For example, a “<*>” can be either “00:01” or “<1 sec”. This means that a “<*>ïs not necessarily a token, but a number of tokens. This type of dynamic variable has an impact on our parsing accuracy, and we next wanted to do further research on log datasets with this type of dynamic variable.

REFERENCES

[1] L. Tang, T. Li, and C.-S. Perng, “LogSig: Generating system events from raw textual logs,” in CIKM, 2011, pp. 785–794.

[2] Tao Li, Yexi Jiang, Chunqiu Zeng, Bin Xia, Zheng Liu, Wubai Zhou, et al, “Flap: An end-to-end event log analysis platform for system management,” in Proceedings of the 23rd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, 2017, pp. 1547–1556.

[3] S. Khatuya, N. Ganguly, J. Basak, M. Bharde and B. Mitra, “ADELE: anomaly detection from event log empiricism,” IEEE INFOCOM 2018 - IEEE Conference on Computer Communications, 2018, pp. 2114-2122.

[4] T. Barik, R. DeLine, S. Drucker and D. Fisher, “The bones of the system: A case study of logging and telemetry at Microsoft,” 2016 IEEE/ACM 38th International Conference on Software Engineering Companion (ICSE-C), 2016, pp. 92-101.

[5] Wei Xu, Ling Huang, Armando Fox, David Patterson, and Michael I. Jordan, “Detecting large-scale system problems by mining console logs,” in Proceedings of the ACM SIGOPS 22nd symposium on Operating systems principles (SOSP ’09), 2009, pp. 117–132.

[6] Tatsuaki Kimura; Keisuke Ishibashi; Tsuyoshi Mori; Hiroshi Sawada; Tsuyoshi Toyono; Ken Nishimatsu, et al, “Spatio-temporal factorization of log data for understanding network events,” IEEE INFOCOM 2014 - IEEE Conference on Computer Communications, 2014, pp. 610-618.

[7] D. Yuan, H. Mai, W. Xiong, L. Tan, Y. Zhou, and S. Pasupa-thy, “Sherlog: Error diagnosis by connecting clues from run-time logs,” SIGARCH Comput. Archit. News, vol. 38, no. 1, pp. 143–154, Mar. 2010.

[8] C. H. Kim, J. Rhee, H. Zhang, N. Arora, G. Jiang, X. Zhang, and D. Xu, “Introperf: Transparent context-sensitive multilayer performance inference using system stack traces,” SIGMETRICS Perform. Eval. Rev., vol. 42, no. 1, pp. 235–247, Jun. 2014.

[9] K. Zhang, J. Xu, M. R. Min, G. Jiang, K. Pelechrinis and H. Zhang, “Automated IT system failure prediction: A deep learning approach,” 2016 IEEE International Conference on Big Data (Big Data), 2016, pp. 1291-1300.

[10] P. He, J. Zhu, Z. Zheng and M. R. Lyu, “Drain: An online log parsing approach with fixed depth tree,” 2017 IEEE International Conference on Web Services (ICWS), 2017, pp. 33-40.

[11] Makanju, A.A., Zincir-Heywood, A.N., Milios, E.E., “Clustering event logs using iterative partitioning,” in Proceedings of the 15th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, 2009, pp. 1255–1264.

[12] R. Vaarandi and M. Pihelgas, “LogCluster - A Data Clustering and Pattern Mining Algorithm for Event Logs,” in Proceedings of the 11th International Conference on Network and Service Management (CNSM 2015), 2015, pp. 1–7.

[13] P. He, J. Zhu, S. He, J. Li and M. R. Lyu, “An evaluation study on log parsing and its use in log mining,” 2016 46th Annual IEEE/IFIP International Conference on Dependable Systems and Networks (DSN), 2016, pp. 654-661.

[14] 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), 2019, pp. 121–130.

[15] A. Oliner and J. Stearley, “What Supercomputers Say: A Study of Five System Logs,” 37th Annual IEEE/IFIP International Conference on Dependable Systems and Networks (DSN’07), 2007, pp. 575-584.

[16] B. Schroeder and G. A. Gibson, “Disk failures in the real world: What does an MTTF of 1, 000, 000 hours mean to you?” in 5th USENIX Conference on File and Storage Technologies, FAST 2007, February 13-16, 2007, San Jose, CA, USA, 2007, pp. 1–16.

[17] H. Mi, H. Wang, Y. Zhou, R. Lyu, and H. Cai, “Toward fine-grained, unsupervised, scalable performance diagnosis for production cloud computing systems,” IEEE Transactions on Parallel and Distributed Systems, vol. 24, pp. 1245–1255, 2013.

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

[19] Wei Xu, Ling Huang, Armando Fox, David Patterson, and Michael Jordan, “Largescale system problem detection by mining console logs,” in Proceedings of SOSP’09, 2009.

[20] W. Xu, L. Huang, and M. I. Jordan, “Experience mining google’s production console logs.” in SLAML, 2010.

[21] Z. M. Jiang, A. E. Hassan, P. Flora and G. Hamann, “Abstracting Execution Logs to Execution Events for Enterprise Applications (Short Paper),” 2008 The Eighth International Conference on Quality Software, 2008, pp. 181-186.

[22] H. Dai, H. Li, C. S. Chen, W. Shang and T. Chen, “Logram: Efficient log parsing using n-Gram dictionaries,” in IEEE Transactions on Software Engineering, 2020.

[23] 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.

[24] 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.

[25] S. Messaoudi, A. Panichella, D. Bianculli, L. Briand, and R. Sasnauskas, “A search-based approach for accurate identification of log message formats,” in ICPC, 2018.

[26] M. Mizutani, “Incremental mining of system log format,” in SCC, 2013, pp. 595–602.

[27] A. Makanju, A. N. Zincir-Heywood and E. E. Milios, “A Lightweight Algorithm for Message Type Extraction in System Application Logs,” in IEEE Transactions on Knowledge and Data Engineering, vol. 24, no. 11, pp. 1921-1936, Nov. 2012.

[28] D’Souza SC, “LSTM neural network for textual ngrams,” https://doi.org/10.7287/peerj.preprints.27377v1, 2018.

[29] He Shilin, Zhu Jieming, He Pinjia and Michael R. Lyu, “Loghub: A Large Collection of System Log Datasets towards Automated Log Analytics,” arXiv preprint arXiv: 2008.06448, 2020.

[30] Sasho Nedelkoski, Jasmin Bogatinovski, Alexander Acker, Jorge Cardoso, Odej Kao. Self-supervised Log Parsing.arXiv preprint arXiv: 1907.12412, 2019.