METING: A Robust Log Parser Based on Frequent n-Gram Mining

Oihana Coustié, Josiane Mothe, Olivier Teste, Xavier Baril

METING: A Robust Log Parser Based on Frequent n-Gram Mining

Oihana Coustié, Josiane Mothe, Olivier Teste
IRIT UMR5505 CNRS, Université de Toulouse
Toulouse, France
Email: firstname.lastname@irit.fr

Xavier Baril
Airbus
Toulouse, France
Email: firstname.lastname@airbus.com

***Abstract*—Execution logs are a pervasive resource to monitor modern information systems. Due to the lack of structure in raw log datasets, log parsing methods are used to automatically retrieve the structure of logs and gather logs of common templates. Parametric log parser are commonly preferred since they can modulate their behaviour to fit different types of datasets. These methods rely on strong syntactic assumptions on log structure e.g. all logs of a common template have the same number of words. Yet, some reference datasets do not comply with these assumptions and are still not effectively treated by any of the state-of-the-art log parsers. We propose a new parametric log parser based on frequent n-gram mining: this soft text-driven approach offers a more flexible syntactic representation of logs, which fits a great majority of log data, especially the challenging ones. Our comprehensive evaluations show that the approach is robust and clearly outperforms existing methods on these challenging datasets.**

***Index Terms*—log parsing; log data; web service management;**

I. Introduction

Execution logs constitute a pervasive resource for web service management. Logs are recognized to be systematically-available resources and to contain valuable run-time information [1]. They are intensively used by service providers for anomaly detection or performance analysis [2], [3], and by service users for business process mining [4] or user behaviour analysis [5].

A *log* can be defined as a semi-structured message, automatically generated through a `print` command, and traces the system execution. It contains valuable monitoring information, such as the *timestamp* (time when the log occurred), the *level* (the severity of the event), or the *content*, an unstructured free text. The latter often follows a determined *template*, induced by the generative code. The *event type* of a log denotes the type of information being logged, and gathers logs with common templates. The event type is a crucial information since most log mining methods opt for a data representation based on the event type [6], [7]. Unfortunately, the content part of the logs do not contain any structural information and the event type is often unavailable [8]. Therefore, there is a substantial need for automated solutions to structure these messages and retrieve their event types.

Log parsers are automated data-driven solutions to infer the event types of logs [9]. They create groups of similar logs and deduce the underlying structure of the retrieved groups. The existence of an underlying structure and the high volumes of log datasets motivate log parsers to exclusively rely on syntactic analysis, discarding any semantic aspect. Yet, the meaning of log statements influences the labelling of log groups, introducing important syntactic heterogeneity among the groups' profiles. The ability of a log parser to modulate its behaviour to different grouping profiles is therefore key to insure its robustness. To achieve this modulation, most of the existing log parsers [10], [11] are parametric : they rely on hyper-parameters which can be tuned to propose different versions of parsing groups. The results of the benchmark of Zhu *et al.* [8] show the superiority both in accuracy and robustness of these parametric methods over non-parametric ones [12]. Hence, the robustness of a log parser across different types of log data depends on its ability to modulate, which is enhanced by the use of hyper-parameters.

The literature contains only few examples of such adaptive log parsers [10], [11], [13]. These modern techniques rely on strong syntactic assumptions; e.g. a group can only contain logs of the same *length* (number of words) [11], [13]. Obviously, these assumptions do not comply with some real-life datasets: Zhu *et al.*' study [8] points out that some reference datasets are challenging to parse because they do not comply with the strong syntactic assumptions of the various state-of-the-art methods. As a result, these datasets do not benefit from any adaptive state-of-the-art log parsing solution, while they include important log data, such as OpenStack, a reference for the evaluation of anomaly detection [2], [3].

Finally, frequent pattern mining methods [14], [15], [16] offer a more text-driven representation. These methods are based on the soft and popular assumption that frequent patterns are likely to be fix parts of the template and offer interesting perspective to enhance flexibility around the syntax. Nonetheless, the existing frequent pattern mining techniques are not robust and show very heterogeneous results [8] either because they are parameter-free — and lack of modulation power — [16] or because their functioning around the frequent patterns still rely on strong assumptions; e.g. frequent tokens always appear at the same position in the log [14].

To solve the challenge of robustness across datasets, we introduce a new parametric log parser, named METI*N*G (Modular Event Type Inference with N-Grams). METI*N*G is modular, thanks to its two hyper-parameters, and uses the

frequent pattern mining approach in a new flexible way: it extracts frequent n-grams, without any additional syntactic assumptions. With a comprehensive evaluation on 16 datasets, we show that METI*N*G is robust and globally outperforms the state-of-the-art methods. METI*N*G is also able to tackle challenging datasets, on which existing methods fail, with some impressive accuracy improvements. We detail our log parser in section II, before evaluating METI*N*G in comparison to the state-of-the-art methods in section III.

II. The METI*N*G parsing method

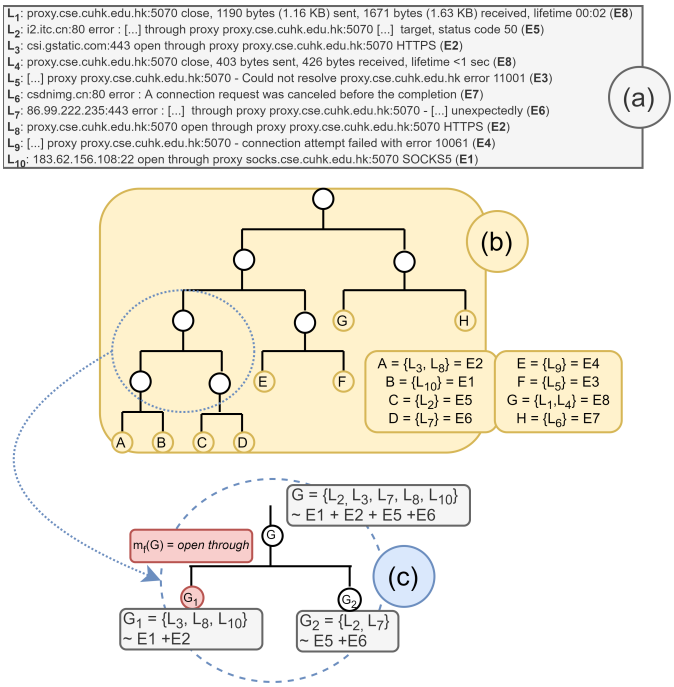

Fig. 1: METI*N*G dendrogram for the Proxifier dataset (a) the original logs (with labels (E_n)) (b) the matching dendrogram with the final groups, perfectly matching the labels (bottom right corner) (c) an intermediate step with G divided in G_1 and G_2, based on the presence of the 2-gram $m_f(G)$

METI*N*G is a parametric log parser that relies on frequent pattern mining through the search of fix parts among frequent n-grams. We first provide an overview of the method before formally defining its functioning and its hyper-parameters.

a) Method overview: Log parsers aim at grouping the logs of a datset. They are evaluated on their ability to retrieve the original *groups of logs*, induced by the labels. In Figure 1(a), the log L_1 is associated to the group E_8 and shall be gathered with the log L_4, also labeled as E_8. METI*N*G generates these groups through the creation of a dendrogram, a tree-like structure, inspired from the hierarchical clustering techniques [17]. Starting from a group with all the logs, our method recursively splits the groups into two sub-groups, as shown in Figure 1(b). When the dendrogram is built, the final groups are retrieved (represented by the capital letters in Figure 1(b)). The dendrogram representation establishes a hierarchy among the groups, and therefore, among the distances between logs. In Figure 1(b) the logs of `A` have a higher similarity with the logs of `B` than they do with those of `F`. This hierarchy fits the nature of textual data: alternative parsing results can be retrieved thanks to dendrogram cuts. In the example, `A` and `B` could be gathered in a larger unique group, with a more generic template. This convenience in providing alternative results complies with the enhancement of modulation power. Furthermore, at each separation step, METI*N*G performs its parsing according to frequent n-gram mining. We assume that logs generated by the same command are likely to have common word sequences, and that frequent n-grams might be the fix parts of group templates. The following paragraphs formally describe the method functioning.

b) Notations: We use parentheses notation $(a_1..a_k)$ to designate an ordered sequence of length k, while the curly brackets $\{a_1, .., a_k\}$ denotes a set of k unordered and unique elements. We add the following nomenclature: a sequence name is noted (a), while a set is named by a capital letter.

c) Problem statement: Let L be a dataset of logs. A log $\ell \in L$ of length $|\ell|$ is composed of the words $w_i^\ell, i < |\ell|$. We note (w_i^ℓ) the sequence of these words and W_ℓ the corresponding set of unique words. When L is labelled, each log is associated to an event type, and all the logs with the same event type form a group. In Figure 1(a), the log L_1 belongs to the group E_8. These groups form a partition of L, noted E_L. In the example of Figure 1(a), $E_L = \{E_1, E_2 ... E_8\}$. A log parser α also creates a partition E_α, by associating a group to each log. In the example, L_1 is associated to the group G and the partition E_α is the set of groups $\{A, B, ...F\}$. The log parsing problem consists in maximizing an accuracy score between E_L and E_α.

d) Group splitting: To create the partition E_α, METI*N*G recursively splits the logs. At each recursive step, illustrated in Figure 1(c), a group of logs $G \subseteq L$ is divided into two disjoint sub-groups $G_1, G_2 \subset G$, with $G_1 \cap G_2 = \emptyset$. This split is based on the frequency of the n-grams in G, where n is a hyper-parameter. For $n \leq |\ell|$, an n-gram of ℓ is a n-long sub-sequence of consecutive words. In Figure 1(a), the log L_2 contains, among others, the 2-grams "open through" and "through proxy". All the n-grams of ℓ constitute its set of n-grams:

$$N_\ell = \{(w_1..w_{n-1})..(w_{k-n+1}..w_k)\}$$

The n-grams of a group G form the set $N_G = \bigcup_{\ell \in G} N_\ell$. For each n-gram $m \in N_G$, its frequency in G is the number of logs of G in which it appears. We define the frequency function by:

$$\begin{array}{rccc} \texttt{freq}_G & : & N_G & \rightarrow & \mathbb{N}^* \\ & & m & \mapsto & |\{\ell \in G \mid m \in N_\ell\}| \end{array}$$

In Figure 1(c), in the presented group `G`$=\{L_2, L_3, L_7, L_8, L_{10}\}$, the 2-gram "open through" appears in L_3, L_8 and L_{10}, so $\texttt{freq}_G(\text{"open through"}) = 3$. We propose to split G according to the presence of the most frequent n-gram in G, $m_f(G) \in N_G$: the logs containing

$m_f(G)$ are gathered in G_1 and the others in G_2. Formally, $G_1 = \{\ell \in G | m_f(G) \in N_\ell\}$, and $G_2 = G - G_1$. Since $m_f(G)$ is the most frequent n-gram in G, it is the most likely to be a fix part. Splitting according to $m_f(G)$ therefore avoids the gathering of logs that share common variable parts (*e.g.* numbers). However, if $m_f(G)$ occurs in every logs of G, then all the logs will be gathered in G_1, preventing the division. Hence, instead of searching $m_f(G)$ in N_G, we rather restrict the research to $N_G^* = \{m \in N_G \mid \texttt{freq}_G(m) < |G|\}$. Finally, the splitting n-gram $m_f(G)$ is defined as:

$$m_f(G) = \underset{m \in N_G^*}{\arg\max} \texttt{freq}_G(m)$$

e) Stopping criteria: We define a condition so as to stop the splitting iteration and render the final groups of logs. This condition is based on the improvement of a homogeneity score, $\texttt{score}(G)$, calculated for each group G: the splitting process is stopped iff $\texttt{score}(G) > \mu \cdot \texttt{score}(G_1)$, where μ is a coefficient of homogeneity improvement. We now define both the homogeneity score $\texttt{score}(G)$ and μ, the coefficient of homogeneity improvement.

We define the homogeneity score as the arithmetical mean of two indicators of homogeneity:

$$\texttt{score}(G) = \frac{\texttt{fix\_word}(G) + \texttt{length\_stability}(G)}{2}$$

First, $\texttt{fix\_word}(G)$ measures the homogeneity of G as the proportion of words appearing in all the logs of G:

$$\texttt{fix\_word}(G) = \frac{|\{w_j \mid \forall \ell \in G, w_j \in (w_i^\ell)\}|}{|\{w_j \in (w_i^\ell) \mid \ell \in G\}|}$$

The numerator calculates the number of words present in all the logs $\ell \in G$ and the denominator counts the total number words in the logs. This indicator associates a high homogeneity score to groups containing numerous common words.

Secondly, the `length_stability` of a group measures the variations among the lengths of logs, $X_G = (|\ell|,\ \ell \in G)$. We denote the p-percentile of X as $Q_p(X)$[1] and define:

$$\texttt{length\_stability}(G) = 1 - \frac{Q_{95}(X_G) - Q_5(X_G)}{Q_{95}(X_G)}$$

The ratio in this formula is equivalent to a relative interquartile range of the lengths distribution. This indicator attributes a high homogeneity score to the groups that have a small variability in their log lengths.

Finally, we define μ, the coefficient of homogeneity improvement, as the arithmetical mean of three indicators:

$$\mu = \frac{1}{3}\left(\frac{1}{ht} + \frac{1}{\texttt{nb\_selection}(G)} + \texttt{freq}_G(m_f(G))\right)$$

$\texttt{nb\_selection}(G)$ is the number of times, during its divisive creation process, that G was generated as a G_1 of some other group — or the number of left branches in the path-tree of G. In Figure 1(b), `B` was selected twice while `F` was only selected once. We observed that groups resulting from a lot of selections are far more homogeneous than unselected groups, since their logs share more common n-grams, by construction. This rate aims at promoting the split of unselected groups, rather than already-selected ones. The rate μ also depends on $\texttt{freq}_G(m_f(G))$. The assumption is that if the most frequent n-gram $m_f(G)$ of a group G has a low frequency, then it is likely that the final group is reached. For instance, if $m_f(G)$ contains a number, it may have a low frequency in the group, and splitting G according to $m_f(G)$ is not desirable. Finally, ht, the *homogeneity tolerance* is a hyper-parameter, described in the following paragraph.

f) Hyper-parameter description: As aforementioned, our method censuses two hyper-parameters:

- n is the length of the n-grams. We mention the specific case, for a log ℓ where $n > |\ell|$. Therefore, ℓ does not contain any n-gram. Hence, the "short" logs of a dataset are parsed apart from the others, thanks to 1-grams. n should be carefully chosen so as to avoid splitting the logs of a common group between the two categories. On the contrary, this n-length threshold can promote a difficult separation between two similar groups of logs.
- ht, the homogeneity tolerance is a rate, with $0 < ht \leq 1$, that controls how divisive the algorithm is. High value of ht implies small values of μ, leading to strict splitting conditions, preventing the algorithm to be divisive. ht enables the tuning of the algorithm divisibility.

III. Evaluation

This section evaluates the performance of METI*N*G on datasets from different sources and compare it to the state-of-the-art references. We also conduce a failure analysis on the various algorithms.

A. Parsing evaluation framework

Our evaluation relies on the complete framework of `Logpai` [18], which evaluates 13 methods on 16 datasets.

1) Experimental protocol and evaluation measures: To evaluate the methods, we perform the log parsing of the 16 datasets of `LogPai`. We first preprocess the logs (as proposed in `LogPai`), apply the parsing method, retrieve the corresponding event types, and compare them to the manually-obtained labels. A good parsing algorithm gathers logs similarly to the groups induced from the labels. External measures (e.g. precision, recall and F-measure) are often employed to calculate pairwise relationship matches [12], [13]. However, as explained by Zhu *et al.*, these indicators tend to provide very smoothed positive results [8], due to a great number of true negatives (logs that are successfully not gathered). The authors define the more rigorous measure of parsing accuracy as the ratio of logs that are exactly correctly parsed. If a log sequence with ground-truth labels [`E1`, `E2`, `E2`] is parsed to [`E1`, `E2`, `E3`], the accuracy is 1/3 since the first log is correctly parsed alone while the two others are incorrectly separated. We run the parametric methods several times in a grid-search way and retrieve the best accuracy result recorded over 120 combinations for each log parser and each dataset.

[1] $Q_p(X)$ is the value so that p% of the values of X are inferior to $Q_p(X)$

TABLE I: Data description (size = number of groups) and accuracy results (as defined in III-A1) of the methods on 16 datasets. Rank: the rank of METI*N*G compared to the other methods. M: mean accuracy results over the datasets. *D*: the mean distance to best accuracy results (see III-B)

Dataset[1]	Mac	Win	Lin	OS	HD	ZK	Sp	H	TB	BGL	HPC	An	HA	OH	Ap	Pr	M	*D*
Data description																		
Source [2]	Operating sys.			Distributed sys.					Super computer			Mobile sys.		Server app.		Sw		
Size	341	50	118	43	14	50	36	114	149	120	46	166	75	37	6	8	86	
Accuracy results																		
Drain	**85,9**	**99,7**	69,0	88,1	99,8	**98,8**	92,0	94,8	**95,8**	**97,3**	90,1	91,3	78,0	78,8	**100**	52,7	88,3	7,8
IPLoM	67,3	68,4	67,6	87,1	**100**	98,4	92,0	95,5	66,3	94,4	82,9	71,2	89,0	**87,1**	**100**	51,7	82,4	13,6
Spell	75,7	98,9	63,9	80,6	**100**	96,4	91,9	77,8	93,4	78,7	65,4	**91,9**	63,9	55,4	**100**	52,7	80,4	15,7
MoLFI	63,6	40,6	28,4	21,3	99,8	83,9	41,8	**95,7**	64,6	96,0	82,4	78,8	44,0	50,0	**100**	1,3	62,0	34,1
Logram	74,4	97,4	20,1	24,6	93,0	95,5	91,6	92,0	55,4	80,5	**97,8**	67,4	**98,1**	55,6	**100**	50,4	74,6	21,5
METI*N*G	82,4	99,6	**92,2**	**96,9**	**100**	96,5	**99,6**	91,1	93,1	88,9	91,8	91,1	68,8	55,5	**100**	**100**	**90,5**	**5,6**
Rank	2	2	1	1	1	3	1	5	3	4	2	3	4	4	1	1		

2) State-of-the-art methods selection: Due to space constraints, we select the most promising methods among the state-of-the-art algorithms, according to the first results of `Logpai`'s evaluations [8], namely Drain[11], IPLoM[13] and Spell[10]. To assess our proposition on the influence of hyper-parameters on parsing results, we also select the recent parameter-free algorithms MoLFI[12] and Logram[16]. By selecting a fewer set of methods, we go further into the analysis of results, deciphering the typical errors and weaknesses of the methods.

3) Data presentation: The proposed log datasets were made available with the `Logpai` tool. The framework gathers 16 datasets composed of 2000 log lines each, and manually labelled. As shown in Table I, the different datasets present an interesting variety both in term of structure (number of groups: Size line in Table I) and on generation source types (Source line in Table I).

B. Parsing results

Table I presents the best accuracy results obtained by a grid-search optimization for the selected algorithms on the datasets. We also mention the rank our method obtains. Finally, we calculate the mean distance to the best scores for each method D. For each method, column D is the difference between its mean accuracy score and the mean accuracy of the best results.

We observe that our log parser, METI*N*G achieves overall better results than the state-of-the-art propositions, with some impressive improvements compared to the existing results, e.g. +47.3% for Pr (Proxifier), +22.3% for Lin (Linux) and +8.8% for OS (OpenStack). These datasets are difficult to parse for most of the existing log parsers: even the most promising ones – namely Drain, IPLoM and Spell – show limited performance on these log datasets. On the contrary, METI*N*G is robust and is the only method that provides an effective solution for these challenging datasets. METI*N*G is ranked best on 6 datasets over the 16, and second on 3. Moreover, METI*N*G has a very low mean distance to the best scores. That means that choosing METI*N*G as a unique solution – instead of selecting the best method for each dataset individually – induces a small loss in performance (only 5.6% on average). In comparison the second best method, Drain, generates a loss of 7.8% and the worst one, MoLFI, a loss of 34.1%. METI*N*G is therefore a more reliable solution than its competitors.

[1]Abbreviations: Win=Windows, Lin=Linux, OS=OpenStack, HD=HDFS, ZK=Zookeeper, Sp=Spark, H=Hadoop, TB=Thunderbird, An=Android, HA=HealthApp, OH=OpenSSH, Ap=Apache, Pr=Proxifier

[2]Abbreviations: sys=system, app=application, Sw= Standalone Software

C. Error analysis through parameter requirements

We analyse the parsing errors made by the methods. Rather than simply censusing the splitting errors, we emphasize and explain some typical and unavoidable mistakes of the different methods. In a recommendation purpose, we focus on the more accurate and robust parametric methods, namely Drain, Spell, IPLoM and METI*N*G.

1) Logs of different lengths: The framework contains examples of groups gathering logs of different lengths (`E8` in Pr, `E146` in TB). Since Drain and IPLoM rely on the assumption that logs of a same group necessarily have the same length, they fail in parsing these datasets. In turn, Spell and METI*N*G opt for a more flexible representation of logs that allows to group logs of different lengths.

2) Variable first words: Drain assumes that the fix tokens of a log are positioned at the beginning, and imposes d common first words in each group, where d is a hyper-parameter. Yet, some groups violate this assumption, like in the OS dataset, with the group `E11`:

> [instance: 54b44eb-2d1a-4aa2-ba6b-074d35f8f12c] Terminating instance AND [instance: 17288ea8-cbf4-4f0e-94fe-853fd2735f29] Terminating instance

Since these logs only have one first word in common, d must be set to 1. Otherwise, the two logs would be separated. However, this strong constraint prevents the separation of other groups of the dataset (`E22` and `E20`). This example traduces a problem of parameter optimization, with the necessity to perform an arbitrage. METI*N*G succeeds in parsing this dataset since it does not impose any constraint on the positions of fix and variables words.

3) Template inclusion: In some datasets, the template of a group is included in the template of another, such as in OH with the groups `E19` and `E20`, defined by the patterns:

E19 = pam_unix(sshd:auth): authentication failure; logname= uid=0 euid=0 tty=ssh ruser= rhost=.*
E20 = pam_unix(sshd:auth): authentication failure; logname= uid=0 euid=0 tty=ssh ruser= rhost=.* user=.*

Since the LCS of these two logs is the whole template of E19, Spell regard these logs as very similar and fails in separating them. In METING, the selected n-gram to separate these logs necessarily comes from E20. Yet, in this example, E20 is minority compared to E19, so the splitting n-gram has a very low frequency and triggers the stopping criteria. Drain also fails in splitting this group, since the logs share numerous common first words. IPLoM is not concerned by this issue and succeeds in parsing the dataset.

4) Alternation fix/variable words: Some group patterns in the dataset show an important alternation, with consecutive fix and variable words, such as the group E8 of Pr, presented in logs L_1 and L_4 of Figure 1(a). Spell finds a low LCS rate for these logs and separate them. IPLoM separates them since they have different lengths. Drain would not stand a chance to gather these logs: they have different lengths and start with a variable token. Only METING is flexible enough to retrieve the important similarity of these logs thanks to their common 2-gram "received lifetime". This explains METING's tremendous improvement of accuracy score for the Pr dataset.

5) General observations: Apart from these situations, we observed that IPLoM tends to have a limited division power. Indeed, IPLoM gathers logs from the same length, then splits them only if they have no common words, and performs a last binary split according to its hyper-parameters. Hence, the method sometimes lacks of division opportunities and logs might be incorrectly gathered (E7 and E8 in ZK). In opposition, METING has a dedicated hyper-parameter, ht, which can modulate its division power.

Spell proceeds in a online way and builds the LCS of a group on-the-fly. Hence, it is very dependant on the order in which logs appear, and commits errors when a variable part is chosen as LCS. Drain also suffers from this dependency: its syntactic assumptions are actually used to point the arriving logs to the right groups. Our solution to extend METING in an online fashion cope with these issues thanks to a first offline pass to build a first version of the dendrogram.

IV. Conclusion

In this paper, we proposed METING, a parametric log parser based on frequent n-gram mining. We showed that it performs all the state-of-the-art methods on many of the 16 reference data sets, and is better on average than the reference log parsers. We also analysed some of the failures that the various methods encounter when applied on the different data sets. In future work, we will study the potential of parameter sensitivity to enhance the modulation power and will implement an online extension of our log parser, thanks to its tree structure. We would like to generalize our method in two ways (a) in the task of word stemming for IR [19] (b) for online anomaly detection on parameter values.

References

[1] B. Chen and Z. M. J. Jiang, "Characterizing logging practices in java-based open source software projects - a replication study in apache software foundation," Empirical Software Engineering, vol. 22, no. 1, pp. 330–374, 2017.

[2] M. Du, F. Li, G. Zheng, and V. Srikumar, "Deeplog: Anomaly detection and diagnosis from system logs through deep learning," in ACM SIGSAC Conf. on Computer and Communications Security, B. M. Thuraisingham, D. Evans, T. Malkin, and D. Xu, Eds., 2017, pp. 1285–1298.

[3] X. Baril, O. Coustié, J. Mothe, and O. Teste, "Application performance anomaly detection with LSTM on temporal irregularities in logs," in CIKM '20: The 29th ACM International Conference on Information and Knowledge Management, Virtual Event, Ireland, October 19-23, 2020, M. d'Aquin, S. Dietze, C. Hauff, E. Curry, and P. Cudré-Mauroux, Eds. ACM, 2020, pp. 1961–1964. [Online]. Available: https://doi.org/10.1145/3340531.3412157

[4] H.-J. Cheng and A. Kumar, "Process mining on noisy logs—can log sanitization help to improve performance?" Decision Support Systems, vol. 79, pp. 138–149, 2015.

[5] X. Yu, M. Li, I. Paik, and K. H. Ryu, "Prediction of web user behavior by discovering temporal relational rules from web log data," in Int. Conf. on Database and Expert Systems Applications. Springer, 2012, pp. 31–38.

[6] S. He, J. Zhu, P. He, and M. R. Lyu, "Experience report: System log analysis for anomaly detection," in 27th IEEE International Symposium on Software Reliability Engineering, ISSRE. IEEE Computer Society, 2016, pp. 207–218.

[7] Q. Fu, J. Lou, Y. Wang, and J. Li, "Execution anomaly detection in distributed systems through unstructured log analysis," in ICDM 2009, The Ninth IEEE International Conference on Data Mining, Miami, Florida, USA, 6-9 December 2009, W. Wang, H. Kargupta, S. Ranka, P. S. Yu, and X. Wu, Eds. IEEE Computer Society, 2009, pp. 149–158.

[8] J. Zhu, S. He, J. Liu, P. He, Q. Xie, Z. Zheng, and M. R. Lyu, "Tools and benchmarks for automated log parsing," in International Conference on Software Engineering: Software Engineering in Practice, ICSE (SEIP), H. Sharp and M. Whalen, Eds. IEEE/ACM, 2019, pp. 121–130.

[9] H. Mi, H. Wang, Y. Zhou, M. R. Lyu, and H. Cai, "Toward fine-grained, unsupervised, scalable performance diagnosis for production cloud computing systems," IEEE Trans. Parallel Distrib. Syst., vol. 24, no. 6, pp. 1245–1255, 2013.

[10] M. Du and F. Li, "Spell: Streaming parsing of system event logs," in IEEE 16th International Conference on Data Mining, ICDM, F. Bonchi, J. Domingo-Ferrer, R. Baeza-Yates, Z. Zhou, and X. Wu, Eds. IEEE Computer Society, 2016, pp. 859–864.

[11] P. He, J. Zhu, Z. Zheng, and M. R. Lyu, "Drain: An online log parsing approach with fixed depth tree," in IEEE International Conference on Web Services, ICWS, I. Altintas and S. Chen, Eds. IEEE, 2017, pp. 33–40.

[12] S. Messaoudi, A. Panichella, D. Bianculli, L. C. Briand, and R. Sasnauskas, "A search-based approach for accurate identification of log message formats," in Conference on Program Comprehension, ICPC, F. Khomh, C. K. Roy, and J. Siegmund, Eds. ACM, 2018, pp. 167–177.

[13] A. Makanju, A. N. Zincir-Heywood, and E. E. Milios, "A lightweight algorithm for message type extraction in system application logs," IEEE Trans. Knowl. Data Eng., vol. 24, no. 11, pp. 1921–1936, 2012.

[14] R. Vaarandi, "A data clustering algorithm for mining patterns from event logs," in IEEE Workhop on IP Operations & Management (IPOM). IEEE, 2003, pp. 119–126.

[15] M. Nagappan and M. A. Vouk, "Abstracting log lines to log event types for mining software system logs," in IEEE International Working Conference on Mining Software Repositories, MSR, J. Whitehead and T. Zimmermann, Eds., 2010, pp. 114–117.

[16] H. Dai, H. Li, W. Shang, T. Chen, and C. Chen, "Logram: Efficient log parsing using n-gram dictionaries," CoRR, vol. abs/2001.03038, 2020.

[17] S. C. Johnson, "Hierarchical clustering schemes," Psychometrika, vol. 32, no. 3, pp. 241–254, 1967.

[18] J. Zhu, P. He, S. He, and J. Liu, "Logpai," https://github.com/logpai/logparser, 2019.

[19] X. Baril, O. Coustié, J. Mothe, and O. Teste, "Rfreestem: A multilanguage rule-free stemmer," in Actes du XXXVIIème Congrès

INFORSID, Paris, France, June 11-14, 2019, 2019, pp. 12–29. [Online].
Available: http://inforsid.fr/actes/2019/INFORSID_2019_p12-29.pdf