USTEP: Structuration des logs en flux grâce à un arbre de recherche évolutif

Arthur Vervaet*, Raja Chiky**, Mar Callau-Zori*

*3DS Outscale, Saint-Cloud, France
**ISEP - Institut Supérieur d'Electronique de Paris, Issy-les-Moulineaux, France
arthur.vervaet@outscale.com, raja.chiky@isep.fr

Résumé. Les registres de *logs* enregistrent en temps réel des informations relatives à l'exécution d'un système informatique. Ceux-ci sont régulièrement consultés à des fins de développement ou de surveillance. Afin d'être utilisables plus facilement pour des tâches d'exploration automatique, les messages des *logs* doivent être structurés. Cette étape se déroulant en amont des opérations d'analyses, elle peut devenir un goulot d'étranglement temporel et influencer l'efficacité des méthodes en aval. Dans ce papier, nous présentons USTEP, une méthode de structuration des *logs* en flux. Basée sur un arbre de recherche évolutif, USTEP est capable de découvrir et d'encoder efficacement de nouvelles règles de structuration. Nous présentons ici une évaluation des performances de USTEP sur un panel de 13 jeux de données issus de systèmes différents. Nos résultats mettent en valeur la supériorité de notre approche en matière d'efficacité et de robustesse par rapport à d'autres méthodes de structuration en ligne.

1 Introduction

Pour les services en ligne de grande envergure, une seule anomalie peut impacter des millions d'utilisateurs (Liang et al., 2021). Pouvoir détecter de tels événements en temps réel permet aux équipes de surveillance de limiter leur impact. Les journaux de *logs* enregistrent une vaste gamme d'événements au sein d'un système informatique et sont communément utilisés pour détecter les origines d'une panne, analyser l'activité ou renforcer la sécurité d'une plateforme(Moh et al., 2016). Les *logs* ont prouvé leur valeur pour la détection automatisée d'anomalies dans plusieurs scénarios (Du et al., 2017; Zhang et al., 2019; Meng et al., 2019). Afin de ne pas avoir à traiter des données textes brutes, les méthodes précédemment citées ont recours à une étape de structuration des messages *logs*. Les *logs* et leur message étant produits par des lignes de code spécifiques (*e.g., print(), logger.log()*), le processus de structuration vise à retrouver le motif sous-jacent à chaque entrée (Figure 1).

La qualité de cette étape de structuration a un impact sur l'efficacité des méthodes en aval. La Figure 2 illustre l'influence de la qualité de structuration des logs (*Parsing Accuracy (PA)*) sur la précision (*Anomaly Detection (AD) accuracy*) de Deeplog (Du et al. (2017)) une méthode de détection d'anomalies. Ici, une perte de 0,2 en *PA* réduit de 0,75 la précision *AD* maximale (0,97 à 0,22) et de 0,59 la précision *AD* moyenne (0,78 à 0,19). Le graphique présenté est une distribution des résultats de 10 instances de Deeplog entraînées sur différentes versions

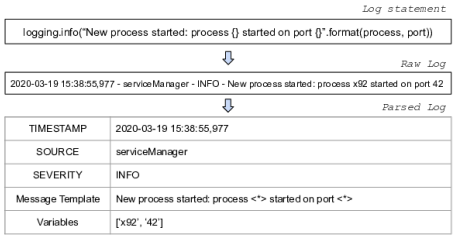


FIG. 1: Exemple de structuration de log

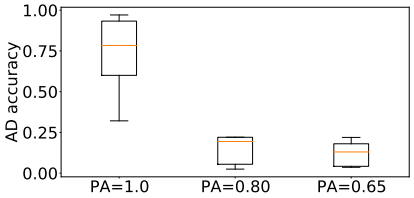


FIG. 2: Précision de Deeplog et qualité de structuration

inégalement structurées d'un jeu de données HDFS [1]. Nous avons utilisé le même ensemble de données et les paramètres décrits dans l'article original présentant Deeplog.

Dans le présent article, nous présentons USTEP, une méthode de structuration en flux des messages *logs* (Section 3). Basée sur un arbre de recherche évolutif, USTEP est capable de traiter des messages en temps constant sans connaissance préalable de l'environnement de *logs*. La Section 4 présente une évaluation comparative de notre approche avec quatre autres méthodes de références. Menée sur 13 ensembles de données issues d'applications différentes, celle-ci met en valeur la supériorité d'USTEP en matière d'efficacité et de robustesse. Les travaux présentés ici ont été réalisés en partenariat avec 3DS OUTSCALE, fournisseur de services *cloud* qui nous a donné accès à des données internes et de la puissance de calcul. Nous avons mis en ligne notre implémentation ainsi que le code des expérimentations avec leurs paramètres afin de permettre une réutilisation facile de notre travail. Les travaux présentés dans ce papier sont un extrait d'une publication pour la conférence internationale en fouille de données IEEE ICDM (Vervaet et al., 2021).

2 Structuration des logs en ligne

L'intérêt de nombreuses applications pour la structuration des *logs* a motivé le développement de plusieurs algorithmes (Zhu et al., 2019). Notre étude de l'état de l'art a permis d'identifier quatre méthodes de structuration en flux : SHISO (Mizutani, 2013) et LenMa (Shima, 2016) basées sur des approches par regroupement de motifs ; Spell (Du et Li, 2016) basée sur une recherche de la plus longue séquence commune entre un message et les patrons déjà découverts ; Drain (He et al., 2017), une méthode utilisant un arbre de partitionnement à profondeur fixe pour encoder des patrons et en découvrir de nouveaux. En raison de leur proximité contextuelle avec la nôtre, ce sont les quatre méthodes que nous avons utilisées dans notre évaluation.

Nous reprenons ici le formalisme introduit par Nedelkoski et al. (2020) et définissons les *logs* comme des séquences de messages textes temporellement ordonnés $\mathcal{L} = (l_i : 1, 2, ...)$, avec i l'index d'un message dans la séquence considérée. Les jetons sont les plus petits éléments insécables d'un message. Chaque message est constitué d'une séquence finie de jetons (mots) séparés par des espaces $\mathbf{t_i} = (t_j^i : t \in \mathcal{T}, j = 1, 2, ..., |\mathbf{t_i}|)$. Avec $\mathcal{T}$ l'ensemble des jetons, j la position d'un jeton au sein d'un message l_i, et $|\mathbf{t_i}|$ le nombre total de jetons dans ce message. L'éclatement en séquence de jetons est une fonction de transformation

1. https ://github.com/logpai/loghub/tree/master/HDFS

$\mathcal{M} : l_i \rightarrow t_i, \forall i$. Ici, $\mathcal{M}$ est une séparation sur les espaces de l_i. Le but des opérations de structuration est de séparer les parties constantes des parties variables d'un message *log*. Nous représenterons ici les patrons et leurs variables par un couple $EV_i = ((e_i, v_i) : e \in \mathsf{E}, i = 1, 2, ...)$. Avec E l'ensemble des patrons et V_i la liste des variables de l_i. Une méthode de structuration de *log* est donc assimilable à une fonction $f : l_i \rightarrow EV_i$.

Avec USTEP, nous proposons une méthode robuste et efficace de structuration des *logs* en flux. Notre travail est motivé par les besoins de l'écosystème *cloud*, cependant nous sommes persuadés qu'un grand nombre de domaines pourrait en profiter (détection d'anomalies (Du et al., 2017), analyse de la consommation (Anitha et Isakki, 2016), dissection d'erreurs (Lu et al., 2017)).

3 USTEP : Algorithme

Une instance d'USTEP est définie par $\mathcal{U} = \{\mathcal{P}, \sigma, \phi\}$. Avec, $\mathcal{P} = \{\mathcal{V}, \widetilde{\mathsf{E}}, \mathcal{E}\}$ un arbre de recherche, $\sigma \in [0; 1]$, et $\phi \in \mathbb{N}^*$ deux paramètres que nous détaillons ultérieurement. Ici $\mathcal{V} = \{v_k\}_{k=1}^{\mathsf{N}}$ représente l'ensemble des nœuds, $\widetilde{\mathsf{E}}$ l'ensemble des patrons découverts, et $\mathcal{E} \subset \mathcal{V} \times \widetilde{\mathsf{E}}$ l'ensemble des liaisons nœuds/patrons.

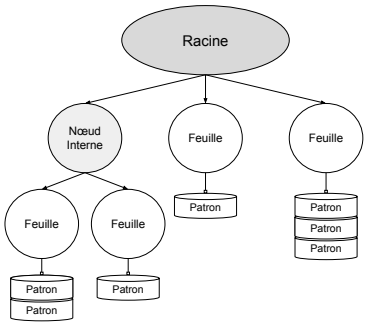


FIG. 3: Structure de l'arbre

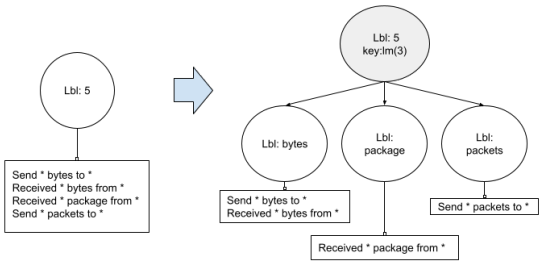


FIG. 4: Exemple : Eclatement de feuille

Structure de l'arbre : USTEP utilise un arbre de recherche évolutif pour encoder les patrons découverts, les raffiner et en découvrir de nouveaux. Celui-ci est constitué de quatre types d'éléments (Figure 3) : une racine v_1, unique nœud présent à l'initialisation ; des nœuds internes servant à encoder les règles de structuration ; des feuilles auxquelles sont attachés des patrons.

Étape préliminaire - Prétraitement : Les utilisateurs peuvent fournir des expressions régulières basées sur leur connaissance du contexte pour améliorer le processus de structuration. Au début de chaque traitement, les jetons concernés seront marqués comme emplacement de variable. Cette étape est optionnelle, mais c'est un moyen d'utiliser sa connaissance du domaine pour améliorer le processus.

Étape 1 - Descente de l'arbre : Les patrons étant attachés aux feuilles, la première étape consiste à atteindre une feuille. La descente d'arbre est opérée par un mécanisme de *key*, *label*. À leur création chaque nœud (interne et feuille) se voit affecter un label. Plusieurs nœuds peuvent avoir le même label, mais celui-ci est unique parmi les enfants d'un même nœud.

Chaque nœud interne a une méthode $key_{v_k}(l_i) \rightarrow t_j$, et la racine une méthode $key_{v_1}(l_i) \rightarrow |\mathbf{t_i}|$. Cette key_{v_1} étant la première règle de descente, elle garantit que tous les patrons attachés à une feuille auront le même nombre de jetons. On commence le processus de descente à la racine ($v = v_1$) et on obtient le nœud suivant grâce à la méthode key du nœud courant v. Cette opération est répétée jusqu'à ce que v soit une feuille. Si aucun nœud fils du nœud courant n'a pour label $key_v(l_i)$, on crée une nouvelle feuille avec ce label, on l'ajoute aux fils du nœud courant.

Étape 2 - Affectation de patron : À la fin de l'étape précédente, une feuille a été atteinte. Ici, on recherche un patron représentatif de l_i. Pour ce faire on calcul le facteur de similarité $simF(l_i, \widetilde{e})$ avec tous les patrons $\widetilde{e} \in \widetilde{\mathsf{E}}$ fils de v. Ce facteur de similarité correspond à la proportion de jetons identiques à même position entre le message l_i et un patron $\widetilde{e}$. Il est calculé comme suit :

$$simF(l_i, \widetilde{e}) = \frac{\sum_{j=1}^{|\mathbf{t_i}|} equ(t_j^i, \widetilde{e}(j))}{|\mathbf{t_i}|} \quad (1) \qquad equ(t_1, t_2) = \begin{cases} 1 & \text{if } t_1 = t_2 \\ 0 & \text{otherwise} \end{cases} \quad (2)$$

Avec $\widetilde{e}(j)$, le j-ème jeton de $\widetilde{e}$. Si la plus grande simF calculée est supérieure à σ, on choisit le patron associé pour représenter l_i. Le cas échéant, un nouveau patron égal à l_i est créé et ajouté à la feuille. Si l_i est assimilé à un patron déjà découvert, celui-ci est mis à jour. Cette mise à jour remplace tous les jetons non identiques à position équivalente par des emplacements de variable ('*').

Étape 3 - Éclatement des feuilles : À la fin de l'Étape 2, un patron $\widetilde{e}$ a été affecté au message log l_i. Afin de perpétuellement augmenter le nombre maximal de calculs de *simF* avec l'augmentation du nombre de patrons découverts : USTEP transforme les feuilles saturées en nœud interne. Une feuille est considérée comme saturée si elle est liée à plus de ϕ patrons. Le processus d'éclatement commence par calculer le nombre de jetons différents à chaque position parmi les patrons de v. La position avec la plus grande diversité est choisie comme pivot p. v est transformé en nœud interne avec une méthode $key(l_i) = t_p^i$. Pour chaque jeton unique parmi les patrons de v à la position p, une nouvelle feuille avec ce label est créée et attachée à v. Les patrons sont alors transférés de v aux nouvelles feuilles en fonction de leurs jetons en position p. La Figure 4 illustre le processus de division pour une feuille saturée avec $\phi = 3$. Dans cet exemple, $p = 3$ car c'est la position avec le plus de jetons différents parmi les patrons (*bytes*, *packages*, *packets*).

Complexité spatiale : USTEP a une complexité spatiale proportionnelle au nombre de patrons et de nœuds dans $\mathcal{P}$, soit $\mathcal{O}(|\mathcal{V}| \times |\widetilde{\mathsf{E}}|)$.

Complexité temporelle : La complexité temporelle d'USTEP est de $\mathcal{O}(|\mathbf{t_i}| - 1 + |\mathbf{t_i}| \times (\phi - 1) + |\mathbf{t_i}| \times (\phi))$, soit $\mathcal{O}(|\mathbf{t_i}|)$. Cela représente le cas ou l'on doit faire $|\mathbf{t_i}| - 1$ étapes de descente d'arbre, suivies de $(\phi - 1)$ calculs de $simF$. Le tout menant à une saturation de la feuille trouvée, ce qui demande $|\mathbf{t_i}| \times \phi$ de plus pour trouver le pivot. En faisant la supposition raisonnable que $mean(|\mathbf{t_i}|)$ est constant dans le temps, on a une complexité temporelle en $\mathcal{O}(1)$. Cette hypothèse est raisonnable puisque la longueur moyenne des *logs* n'est pas vouée à évoluer avec le fonctionnement du système.

Nom	#messages	taille(B)	#patrons	#j/ligne	%variables
Apache	2 000	168K	6	6.3	23%
BGL	2 000	310K	120	6.3	25%
Hadoop	2 000	376K	114	8.4	37%
HDFS	2 000	282K	14	7.4	45%
HPC	2 000	148K	46	3.5	18%
Mac	2 000	312K	341	9.4	28%
OpenSSH	2 000	220K	27	8.7	28%
OpenStack	2 000	582K	43	9.0	45%
Thunderbird	2 000	318K	149	8.5	16%
Zookeeper	2 000	274K	50	6.3	18%
HDFS-2	11 175 629	1.5G	30	7.4	N.A
OpenStack-2	207 820	54M	43	9.0	N.A
Internal-1	1 750 916	1.2G	N.A	28.2	N.A

TAB. 1: Caractéristiques des jeux de données

4 Evaluation

Jeux de données : Afin d'évaluer notre approche, nous avons sélectionné 13 jeux de données provenant d'applications différentes. On retrouvera parmi eux des *logs* venant de systèmes distribués (*HDFS, OpenStack, Zookeeper, Internal-1*), d'applications serveur (*OpenSSH*), de supercalculateurs (*BGL, HPC, Thunderbird*) et d'opérateurs système (*Mac*). À l'exception d'Internal-1 qui est issu de systèmes internes 3DS OUTSCALE, tous ces jeux sont disponibles en ligne. Leur origine et leur méthode de collection sont détaillées dans les travaux de He et al. (2020). Le Tableau 1 regroupe les caractéristiques de ces différents jeux de données avec : le nombre de lignes de *logs* (*#messages*) ; la taille mémoire ; le nombre réel de patrons sous-jacents (*#patrons*) ; le nombre moyen de jetons par ligne (*j/ligne*) et la proportion de jetons variables dans le jeu (*%variables*).

Contexte expérimental : Nous comparons ici les performances d'USTEP avec celles de quatre autres méthodes de structuration des logs en lignes identifiées dans notre état de l'art (Drain, LenMa, SHISO, Spell). Par mesure d'équité, les mêmes expressions régulières ont été appliquées pour le prétraitement de chaque méthode. Les expériences ont été réalisées sur une machine virtuelle *cloud* de type CentOS Linux 7.8 avec 32 cœurs et 62 GB de RAM. Par manque de place, nous ne détaillons pas ici les valeurs utilisées pour les hyperparamètres de chaque méthode, mais ceux-ci sont disponibles en ligne avec le code source des expériences.

Efficacité de la structuration : Les méthodes de structuration des *logs* sont usuellement évaluées grâce à la précision de structuration (PA) (Zhu et al. (2019); Du et Li (2016)). Celle-ci représente le ratio de messages correctement structurés par rapport au nombre total de messages. Un message est considéré comme correctement structuré s'il est associé au même groupe que son patron sous-jacent.

Pour évaluer l'efficacité des méthodes sélectionnées, nous avons utilisé nos 10 jeux labélisés. Afin d'éviter de possible biais, les hyperparamètres de chaque méthode ont été sélectionnés en répétant 100 fois l'expérience avec des valeurs différentes et en conservant le meilleur résultat (Tableau 2). Notre méthode USTEP obtient la meilleure PA moyenne à 0.937 PA et la

Jeu de donnée	Drain	LenMa	SHISO	Spell	USTEP
Apache	**1**	**1**	**1**	**1**	**1**
BGL	0.963	0.690	0.711	0.787	**0.964**
Hadoop	0.948	0.885	0.867	0.778	**0.951**
HDFS	0.998	0.998	0.998	**1**	0.998
HPC	0.887	0.830	0.325	0.654	**0.906**
Mac	0.787	0.698	0.595	0.757	**0.848**
OpenSSH	0.788	0.925	0.619	0.554	**0.996**
OpenStack	0.733	0.743	0.722	**0.764**	**0.764**
Thunderbird	**0.955**	0.943	0.576	0.844	0.954
Z.keeper	0.967	0.841	0.660	0.964	**0.988**
Moyenne	0.903	0.855	0.707	0.810	**0.937**

TAB. 2: Précision des méthodes de structuration

plus haute PA pour 8 des 10 jeux considérés tout en étant très proche du meilleur résultat pour les deux derniers jeux de données.

Robustesse de la structuration : Nous sommes persuadés que pour être utilisée en production, une méthode de structuration de *logs* doit être robuste et maintenir ses performances sur des jeux de données provenant de systèmes divers. La Figure 5 représente une distribution en boite de la PA obtenue par chaque méthode sur les différents jeux de données. Les méthodes sont ordonnées par médiane croissante. SHISO termine en bas de l'échelle et USTEP en haut. La distance du premier au troisième quartile est de 0.23 pour SHISO, 0.146 pour Drain, 0.079 pour Spell et 0.073 pour USTEP. Couplée au fait qu'il ait la meilleure PA moyenne (0.959), la faible variance d'USTEP la rend plus généralisable que les autres méthodes considérées. Cette capacité de généralisation est particulièrement intéressante pour notre contexte *cloud* dans lequel la base de code est soumise à de nombreux changements et où le client est maître des applications qu'il souhaite faire tourner et de leurs versions.

Vitesse de structuration : Structurer les *logs* n'est généralement qu'une étape préliminaire pour des tâches d'inspection et peut donc devenir un goulot d'étranglement temporel. Pour mesurer la capacité de chaque méthode à traiter rapidement des grands volumes de *logs*, nous avons utilisé nos trois jeux de données (*HDFS-2*, *OpenStack-2*, *Internal-1*) et enregistré le temps mis par chaque méthode pour les traiter complètement.

Les trois graphiques de la Figure 7 affichent l'évolution du temps total de traitement relativement au nombre de *logs* traités. Drain et USTEP se présentent comme les méthodes les plus rapides avec un temps de structuration constant. Seules ces deux méthodes ont réussi à traiter *Internal-1* dans un temps avoisinant les 5 heures. Pour ce jeu de données, Spell, SHISO et LenMa n'avaient pas obtenu de résultat en moins d'une semaine. Spell présente une vitesse constante sur *HDFS-2* et *OpenStack* et n'arrive pas à structurer le dernier jeu de données en un temps raisonnable. Plusieurs facteurs expliquent ce résultat : 1) Spell se base sur l'algorithme de recherche de la plus longue séquence commune avec une complexité en $\mathcal{O}(|\mathbf{t_i}| \times m)$, or *Internal-1* a un plus grand nombre moyen de jetons par ligne (28.2) $|\mathbf{t_i}|$ que les autres jeux (7.4 et 9.0) ; 2) *Internal-1* est le seul jeu pour lequel aucune expression régulière n'a été utilisée, ce qui complique la tâche de Spell qui se base sur de nombreuses optimisations pour améliorer sa complexité moyenne.

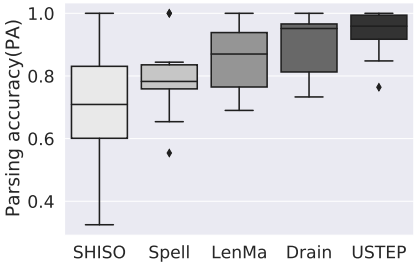
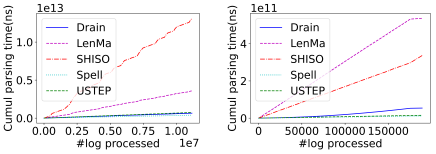
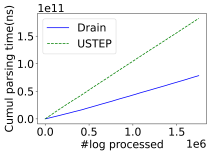


FIG. 5: Robustesse de structuration



(a) HDFS-2 (b) OpenStack-2



(c) Internal-1

FIG. 7: Vitesse de structuration

5 Conclusion

La structuration des messages *logs* est une étape obligatoire pour de nombreuses méthodes d'analyse des logs. Dans cet article, nous avons présenté USTEP, une méthode de structuration en ligne basée sur un arbre de recherche évolutif. Notre évaluation sur 13 jeux provenant d'applications différentes met en valeur le temps de traitement constant ainsi que la supériorité en termes de précision (+3.4% à93.7%) et de robustesse de notre approche par rapport à quatre autres méthodes de l'état de l'art. Dans le futur, nous voulons améliorer la mise à l'échelle de notre méthode au travers d'une étude sur sa distribution sur plusieurs machines. Nous souhaitons également utiliser la sortie d' USTEP pour faire progresser la recherche en détection automatisée des anomalies appliquée aux *logs*.

Remerciements : Nous remercions 3DS OUTSCALE et l'ANRT qui contribuent au financement de nos travaux de recherches au travers d'un programme CIFRE (Convention 2020/0289).

Références

Anitha, V. et P. Isakki (2016). A survey on predicting user behavior based on web server log files in a web usage mining. In *2016 International Conference on Computing Technologies and Intelligent Data Engineering (ICCTIDE'16)*, pp. 1–4. IEEE.

Du, M. et F. Li (2016). Spell : Streaming parsing of system event logs. In *2016 IEEE 16th International Conference on Data Mining (ICDM)*, pp. 859–864. IEEE.

Du, M., F. Li, G. Zheng, et V. Srikumar (2017). Deeplog : Anomaly detection and diagnosis from system logs through deep learning. In *ACM SIGSAC Conference on Computer and Communications Security*.

He, P., J. Zhu, Z. Zheng, et M. R. Lyu (2017). Drain : An online log parsing approach with fixed depth tree. In *2017 IEEE international conference on web services (ICWS)*. IEEE.

He, S., J. Zhu, P. He, et M. R. Lyu (2020). Loghub : A large collection of system log datasets towards automated log analytics. *arXiv preprint arXiv :2008.06448*.

Liang, H., L. Song, J. Wang, L. Guo, X. Li, et J. Liang (2021). Robust unsupervised anomaly detection via multi-time scale dcgans with forgetting mechanism for industrial multivariate time series. *Neurocomputing 423*, 444–462.

Lu, S., B. Rao, X. Wei, B. Tak, L. Wang, et L. Wang (2017). Log-based abnormal task detection and root cause analysis for spark. In *IEEE International Conference on Web Services*.

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

Mizutani, M. (2013). Incremental mining of system log format. In *2013 IEEE International Conference on Services Computing*, pp. 595–602. IEEE.

Moh, M., S. Pininti, S. Doddapaneni, et T.-S. Moh (2016). Detecting web attacks using multi-stage log analysis. In *IEEE 6th international conference on advanced computing (IACC)*.

Nedelkoski, S., J. Bogatinovski, A. Acker, J. Cardoso, et O. Kao (2020). Self-supervised log parsing. *arXiv preprint arXiv :2003.07905*.

Shima, K. (2016). Length matters : Clustering system log messages using length of words. *arXiv preprint arXiv :1611.03213*.

Vervaet, A., R. Chiky, et M. Callau-Zori (2021). Ustep : Unfixed search tree for efficient log parsing. In *2021 IEEE International Conference on Data Mining (ICDM)*.

Zhang, X., Y. Xu, Q. Lin, B. Qiao, H. Zhang, Y. Dang, C. Xie, X. Yang, Q. Cheng, Z. Li, et al. (2019). Robust log-based anomaly detection on unstable log data. In *ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering*.

Zhu, J., S. He, J. Liu, P. He, Q. Xie, Z. Zheng, et M. R. Lyu (2019). Tools and benchmarks for automated log parsing. In *2019 IEEE/ACM 41st International Conference on Software Engineering : Software Engineering in Practice (ICSE-SEIP)*, pp. 121–130. IEEE.

Summary

Logs record valuable system information at runtime. They are widely used by data-driven approaches for development and monitoring purposes. Parsing log messages to structure their format is a classic preliminary step for log-mining tasks. As they appear upstream, parsing operations can become a processing time bottleneck for downstream applications. The quality of parsing also has a direct influence on their efficiency. Here, we propose USTEP, an online log parsing method based on an evolving tree structure. Evaluation results on a wide panel of datasets coming from different real-world systems demonstrate USTEP superiority in terms of both effectiveness and robustness when compared to other online methods.