Automated Log-Based Anomaly Detection within Cloud Computing Infrastructures

Arthur Vervaet

Automated Log-Based Anomaly Detection within Cloud Computing Infrastructures

Arthur Vervaet

A dissertation submitted in fulfilment of the requirements of
the degree of *Docteur en Informatique*
Sorbonne Université

Thèse dirigée par:	
Pr. Raja Chiky	3IL
Thèse encadrée par:	
Dr. Yousra Chabchoub	ISEP - Institut Supérieur d'Electronique de Paris
Dr. Mar Callau-Zori	Outscale, Dassault Systèmes
Rapporteurs:	
Dr. Jean-Marc Pierson	Institut de Recherche en Informatique de Toulouse
Dr. Richard Chbeir	Laboratoire Informatique de l'Université de Pau et des Pays de l'Adour
Membres du jury:	
Dr. Julien Sopena	LIP6
Dr. Sylvain Lefebvre	L@bISEN

Acknowledgements

Pursuing a Ph.D. can be a lonely and challeging activity. During this time I receive tremendous and invaluable support from academic supervisors, colleagues at Outscale and at ISEP, friends and family. I will be forever thankful to them.

First and foremost, I would like to thank my girlfriend Camille for her unconditional love and support. Even when I needed to finish a submission at 12 on new year's eve.

Raja Chiky, Mar Callau-Zori and Yousra Chabchoub, for their trust their wisdom and the countless time they spent advising and supporting me. I could never have dreamed of a better supervising team, thanks again for your dedication!

As a member of Outscale, I had the chance to work alongside kind and inspiring people. Evgeny, Christophe, Aurélien, Théophile, David, Bastien, Nicolas, Gautier, Salma, Nesrine it has been a pleasure to exchange with you and to benefit from your expertise on a countless number of topics. Our coffee breaks will forever remain as an enlightening experience.

Cheers to all my friends from the lab Katia, Guillaume, Abdul, Abir, Nan, Shufan, Mariam, Jade and Maurras. I have been lucky to share time with you.

Thanks to my friends and family, Mom, Dad, Nils and Marion, Erwan, Maxime, Alexandra, Antoine, Louis, Bastien, Victor for your unconditional support. Your support always pushed me to do better.

A very special thank to Franck Gautier, who believed in me and without whom I would never have started this project. You will forever remain a friend and the most inspiring teacher I know.

I also acknowledge the French National Association of Research and Technology (ANRT) for funding this thesis through grant CIFRE N. 2020/0289.

This is for my dad. Thanks for making me love science and much more. Fines non habemus.

Résumé

Les plateformes de *Cloud Computing* mettent à disposition de leurs clients différentes ressources informatiques à la demande. Cette externalisation rend les fournisseurs garants de la haute disponibilité et de la qualité de leurs services. La gestion d'un parc de ressources mutualisées en croissance constante demande de minimiser l'intervention humaine afin de suivre le changement d'échelle des infrastructures et d'éviter les erreurs. Dans cette thèse, réalisée en collaboration avec 3DS OUTSCALE, un fournisseur français de cloud public, nous explorons le potentiel des logs informatiques pour la détection automatique d'anomalies au sein des plateformes de cloud computing.

Les journaux de logs sont écrits pendant l'exécution et fournissent des informations sur l'état actuel d'un système. Ils sont déjà largement utilisés à des fins diverses, telles que la surveillance, le diagnostic, l'évaluation des performances ou la maintenance. Cependant, l'utilisation des logs pour la détection automatique et en temps réel d'anomalies reste compliquée. La nature complexe des plateformes de cloud computing doit être dûment prise en compte. L'extraction d'informations pertinentes à partir d'une multitude de sources de logs et les évolutions fréquentes de la base de code posent des défis et introduisent des risques d'erreurs. De plus, établir des relations entre les logs au sein de tels systèmes est souvent une tâche impossible.

Les solutions de structuration visent à retrouver les variables injectées dans les messages des logs. Notre première contribution implique une étude approfondie de deux de ces méthodes en examinant l'impact de l'optimisation des hyperparamètres et du prétraitement sur leur précision. Étant donné la nature laborieuse de l'étiquetage des logs dans le contexte des plateformes de cloud computing, nous avons cherché à identifier des valeurs génériques potentielles permettant une analyse précise dans divers scénarios. Cependant, nos recherches révèlent l'impossibilité de trouver de telles valeurs, soulignant ainsi la nécessité d'approches de structuration des logs plus robustes.

Notre deuxième contribution présente USTEP, une approche innovante de structuration des logs en ligne qui surpasse les méthodes existantes en termes de précision, d'efficacité et de robustesse. USTEP atteint une complexité temporelle d'analyse constante dans le pire des cas, le distinguant ainsi de ses prédécesseurs pour qui le nombre de patrons déjà découverts ralentit la vitesse de structuration. À travers une analyse comparative de cinq méthodes de structuration en ligne des logs utilisant 13 ensembles de données open source et un ensemble de données dérivé des systèmes de 3DS OUTSCALE, nous démontrons les performances supérieures d'USTEP. De plus, nous proposons USTEP-UP, une architecture qui permet l'exécution distribuée de plusieurs instances d'USTEP.

Notre troisième contribution présente Monilog, une architecture système conçue pour la détection automatique des anomalies à partir de journaux de logs. Monilog exploite des paires modèle/métrique pour prédire l'activité logs au sein d'un système et détecter les anomalies en identifiant des changements de comportement. Les capacités prédictives de Monilog sont

reforcées par notre utilisation des récentes avancées dans le domaine de l'apprentissage automatique. Il génère également des rapports détaillés mettant en évidence les composants impliqués et les applications associées à une anomalie.

Nous avons implémenté une instance de Monilog à l'échelle d'une plateforme cloud et mené des analyses expérimentales pour évaluer sa capacité à prévoir des événements anormaux, tels que des pannes de serveur résultant de problèmes de virtualisation. Les résultats obtenus soutiennent fortement notre hypothèse concernant l'utilité des logs pour la détection et la prévision d'événements anormaux. Notre implémentation de Monilog a identifié avec succès des périodes anormales et fournie des informations précieuses sur les applications concernées.

Avec Monilog, nous démontrons la valeur des logs pour la prédiction des anomalies dans de tels environnements et proposons une architecture flexible pour les études futures. Notre travail dans le domaine de la structuration des logs avec la proposition d'USTEP et d'USTEP-UP nous fournit non seulement des informations supplémentaires pour la construction de modèles de détection des anomalies, mais présente également des avantages potentiels pour d'autres applications d'exploration des logs.

Keywords: *Fouille de Données, Cloud Computing, Détection d'Anomalies, Apprentissage Profond, Flux de Données, Séries temporelles, Systèmes Distribués*

Abstract

Cloud computing aims to optimize resource utilization while accommodating a large user base and elastic services. Within this context, cloud computing platforms bear the responsibility of managing their customers' infrastructure. The management of an ever-expanding number of IT resources poses a significant challenge. In this study, conducted in collaboration with 3DS OUTSCALE, a French public cloud provider, we investigate the potential of log data as a valuable source for automated anomaly detection within cloud computing platforms.

Logs serve as a widely utilized information source for various purposes, including monitoring, diagnosing, performance evaluation, and maintenance. These logs are generated during runtime and provide insights into the current state of a system. However, achieving automated real-time anomaly detection based on log data remains a complex undertaking. The intricate nature of cloud computing platforms must be duly considered. Extracting relevant information from a multitude of logging sources and accounting for frequent code base evolution poses challenges and introduces the potential for errors. Furthermore, establishing log relationships within such systems is often an insurmountable task.

Log parsing solutions aim to extract variables from the template of log messages. Our first contribution involves a comprehensive study of two state-of-the-art log parsing methods, investigating the impact of hyperparameter tuning and preprocessing on their accuracy. Given the laborious nature of labeling logs related to a cloud computing platform, we sought to identify potential generic values that enable accurate parsing across diverse scenarios. However, our research reveals the infeasibility of finding such requirements, thereby emphasizing the necessity for more robust parsing approaches.

Our second contribution introduces USTEP, an innovative online log parsing approach that surpasses existing methods in terms of accuracy, efficiency, and robustness. Notably, USTEP achieves a constant worst-case parsing time complexity, distinguishing it from its predecessors for which the number of already detected templates is to be taken into account. Through a comparative analysis of five online log parsers using 13 open-source datasets and one derived from 3DS OUTSCALE systems, we demonstrate the superior performance of USTEP. Furthermore, we propose USTEP-UP, an architecture that enables the distributed execution of multiple USTEP instances.

Our third contribution presents Monilog, a system architecture designed for automated log-based anomaly detection within log data streams. Monilog leverages model/metric pairs to predict log traffic patterns within a system and detect anomalies by identifying deviations in system behavior. Monilog forecasting models are powered by the recent advances in the deep learning field and is able to generate comprehensive reports that highlight the relevant system components and the associated applications.

We implemented an instance of Monilog at cloud scale and conducted experimental analyses to evaluate its ability to forecast anomalous events, such as servers crashes resulting from

virtualization issues. The results obtained strongly support our hypothesis regarding the utility of logs in detecting and predicting abnormal events. Our Monilog implementation successfully identified abnormal periods and provided valuable insights into the applications involved.

With Monilog, we demonstrate the value of logs in predicting anomalies in such environments and provide a flexible architecture for future study. Our work on the parsing field with the proposal of USTEP and USTEP-UP not only provides us with additional information for building anomaly detection models but also has potential benefits for other log mining applications.

Keywords: *Data Mining, Anomaly Detection, Cloud Computing, Deep Learning, Data Streams, Distributed System, Time Series*

Contents

List of Figures

List of Tables

List of Algorithms

CHAPTER 1

Introduction

Contents

1.1 Context

Cloud computing services offer users on-demand self-service access to a shared pool of IT resources. This model provides clients with significant advantages [2; 3], including elasticity, enabling flexible scaling of computing resources according to their needs. Additionally, it offers broad network access, allowing users to access cloud services from anywhere at any time. Moreover, the pay-by-use model eliminates the requirement for substantial upfront investments in computing resources, physical infrastructure, electricity consumption, and staff for system administration, network management, and database maintenance. These costs are transferred to the service provider.

The cost-effectiveness and flexibility offered by cloud computing have resulted in a widespread adoption among businesses and organizations. However, this shift towards outsourcing also places the responsibility on cloud service providers to ensure high availability and service quality. From the provider's perspective, this exponential growth (with a projected 29.8% increase in the public sector by 2023 [4]) is accompanied by an expanding deployment of infrastructure to manage. To cope with the constant expansion of resources, it becomes crucial to minimize human intervention and adapt to changes in infrastructure scale while mitigating the risk of errors. Consequently, the development of automated monitoring and alerting tools becomes imperative to support administrators [5; 6; 7].

In large-scale online systems such as cloud computing platforms, even a single incident can have a profound impact on millions of users [8; 9; 10]. This impact extends beyond the immediate inconvenience and leads to significant financial consequences [11] as well as performance losses. For example, a four-hour downtime in Amazon Web Services resulted in a staggering loss of $150 million [1]. Consequently, anomaly detection plays a crucial role in the development of secure and reliable platforms. By enabling timely and accurate identification of anomalous events, operational teams can swiftly respond and take measures to mitigate potential losses [12].

[1] https://aws.amazon.com/fr/message/41926/

1.1.1 The Landscape of Cloud Monitoring

Cloud computing platforms are intricate systems comprised of numerous physical components, distributed across multiple data centers. It is not uncommon for a large data center to contain over 100,000 components, such as servers, network equipment, and storage platforms. Monitoring the health of such infrastructure poses a significant challenge for operational teams [13]. Site Reliability Engineers (SREs) often rely on metrics like network latency or Key Performance Indicators (KPIs), such as the number of active virtual machines per region, to carry out their tasks [14]. These metrics enable passive monitoring, where alerts are triggered when values exceed preset thresholds (e.g., when the memory usage of a machine approaches its limit). Additionally, SREs may employ manually defined rules to detect specific undesirable scenarios, such as denial-of-service and distributed denial-of-service attacks, which pose a serious threat to the availability of cloud computing environments [15]. Studies have shown that rule-based approaches can effectively identify the machines responsible for such attacks and ban their IP addresses [16; 17].

The rule-based approaches lean on expert knowledge to identify and encode new undesirable scenarios. For example, they may monitor updates to lists of common vulnerability exposures. However, the inability to detect or adapt to novel attacks or failures becomes increasingly problematic as the system expands [18]. In the context of cloud computing, platform updates and the diversity of behaviors among components amplify the potential attack vectors and the occurrence of false alarms within the system.

1.1.2 An Introduction to Logging

Capturing runtime information is a common practice within software systems [19; 20]. The produced logs describe a vast range of events as well as variations in the monitored system states [21]. Figure 1.1 is a sample of 20 log lines from a High Performance Computing cluster (HPC) at the Los Alamos National Laboratories log file. Each line within a log file is called a log event and is generated by a log statement inside the source of a program. Logs are

```
2568643 node-70 action start 1074119817 1 clusterAddMember  (command 1902)
2570772 node-124 action start 1074123150 1 clusterAddMember  (command 1900)
2571927 node-28 action start 1074125371 1 risBoot  (command 1903)
2572286 node-17 action start 1074126278 1 bootGenvmunix  (command 1903)
2575909 node-162 action start 1074178193 1 boot  (command 1911)
461572 node-116 action start 1145552379 1 wait  (command 4219)
2566692 1897 boot_cmd success 1073991950 1 Command has completed successfully
2614626 2001 boot_cmd success 1074752045 1 Command has completed successfully
441208 Interconnect-0N00 switch_module fan 1143979502 1 Fan speeds ( 3534 3534 3375 **** 3497 3479 )
130987 3504 boot_cmd abort 1111858191 1 Command has been aborted
27385 Interconnect-0N02 switch_module control 1101771720 1 power/control problem
30700 Interconnect-1T00 switch_module bcast-error 1076189965 1 Link error
198942 gige5 gige temperature 1078030594 1 normal
225213 gige7 gige temperature 1078547960 1 warning
225104 gige7 gige temperature 1078536860 1 normal
224786 node-158 node temperature 1078514733 1 ambient=31
2558082 Interconnect-1N01 switch_module error 1073073636 1 Link ok
2557405 Interconnect-1T01 switch_module error 1072965457 1 Link error
2557080 Interconnect-1N01 switch_module error 1072913499 1 Link error
2556365 Interconnect-1N01 switch_module error 1072790564 1 Link in reset
```

Figure 1.1: A Sample of a Log File From an HPC System

semi-structured data, with a header following a fixed format and a message which is left to the discretion of the developers. For instance, the illustrated HPC logs follow the following format: *"LogId Node Component State Time Flag Message."*

Due to the flexibility of their format, logs have been widely adopted in practice and it is reasonable for the equipment of a single data center to generate billions of log lines per day. Although many logs are being collected and stored during the normal operation of a Cloud platform, they are rarely exploited in real-time. It is when a technical issue arises, or a cloud service delivery is interrupted, the collected logs become the most important source of the troubleshooting and tracing efforts by the SRE department [13].

1.2 Research Objectives and Axis Covered by This Manuscript

In this thesis, we explore the automation of the exploitation of logs to detect anomalies and predict incidents within a cloud computing environment. The goal was to design and implement an innovative autonomous log-based Anomaly Detection (AD) system, capable of detecting anomalies on a cloud computing platform with high accuracy and minimal false positives to

reduce false alarms; characterize the anomalies according to properly target the experts able to manage the alerts and provide them with all relevant information; process a large volume of log files online with a short detection time.

This research project is supported by 3DS OUTSCALE, a French provider of multi-sovereign cloud services. The company extensively employs logs to monitor various systems comprising their cloud computing platform, as well as to identify undesirable outcomes like virtual machine crashes and security breaches. Being able to conduct this study within an industrial context has granted us opportunities to gather insights from field experts and validate our proposals using a large-scale, real-world cloud platform.

1.2.1 Elements of Vocabulary

Within this manuscript, we will regularly use the terms: autonomous, robust, efficient, and precise. We, therefore, think it is important to precisely define here the meaning of these terms within the context of this manuscript:

- **Autonomous:** A system will be considered autonomous if it can perform a targeted task for an extended period (months), without the need for human assistance.
- **Robust:** A robust system retains its operational capability despite changes occurring within its environment. Such environmental changes can for instance be the appearance of new log statements inside the source code of a monitored system and with it the appearance of new log messages.
- **Efficient:** The efficiency of a system or an algorithm reflects its ability to process a large volume of events in a short time. In our context, this means being able of processing millions of log lines per minute, this to be able to operate in real-time.
- **Precise:** The Precision of an alerting system is evaluated using Equation 1.1. With *True Positive* the number of events correctly classified as abnormal and *False Positive*, the number of irrelevant alerts. Note, a system can be perfectly precise but still miss

abnormal events. However, a high precision means a low proportion of *False Positive* and therefore few false alarms.

$$Precision = \frac{True\ Positive}{True\ Positive + False\ Positive} \tag{1.1}$$

1.2.2 An Overview of 3DS OUTSCALE Cloud Platform and Logging Activities

As a cloud provider, 3DS OUTSCALE has two key assets: 1/ the hardware infrastructure built in partnership with CISCO, Intel, Netapp and Nvidia; and 2/ a proprietary orchestrator, TINA OS, to manage the infrastructure and allocate resources to users [22]. The cloud platform of the company is designed to be operated in an API-firsts fashion, meaning that the first entry point for client requests is an ensemble of network-accessible endpoints. Incoming requests are forwarded to the cloud orchestrator (TINA OS) that will interact with other services as well as hardware components to perform the appropriate tasks to satisfy customers' demands. Figure 1.2 is a simplified version of the company cloud platform architecture.

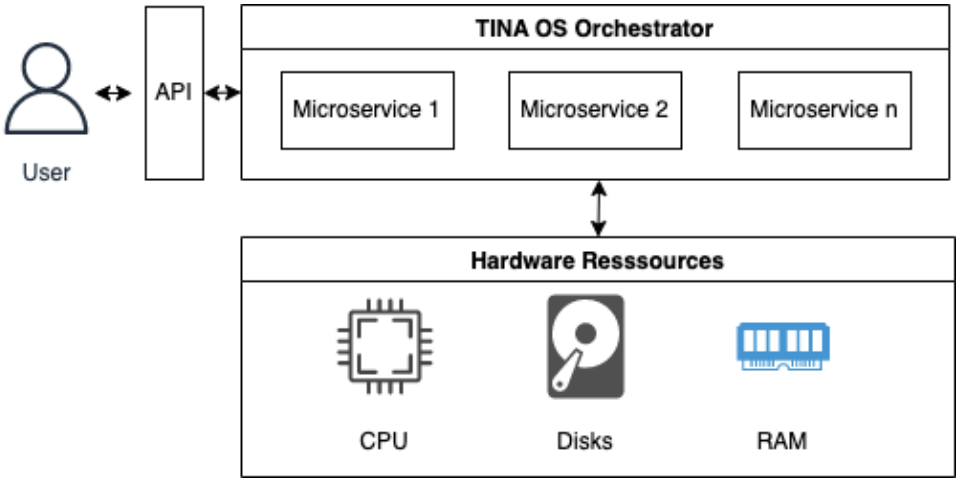


Figure 1.2: 3DS OUTSCALE Simplified Cloud Platform Architecture

From a logging perspective, the software services composing TINA OS, the hardware drivers, and the other services related to the continuity and operations or the monitoring of the platform

are log sources. For the internally developed services of the platforms, teams use continuous integration [23] methods as a delivery model. This is a commonly used practice in software development and a proven way to launch new features quickly. Log entries are produced by logging statements (e.g., print(), logger.log()) inside a program source code, and software version updates often lead to the appearance, modification or deletion of certain types of logs generated by an application. A study conducted by researchers from Microsoft on one of their software shows that after 8 versions, the proportion of changed log statements reaches 30.3% [24].

The challenging aspects of cloud computing platform architectures, a large number of connected devices of various types (thousands inside 3DS OUTSCALE European platform), frequent changes in the code base, and an important logging volumetry (millions of log lines per minute generated by 3DS OUTSCALE European cloud platform) served as the catalyst for the research work presented herein. The main identified challenges revolve around the automated and efficient exploitation of log messages (log parsing), the precise detection of anomalies in highly distributed infrastructure, and the contextualization of the identified anomalous events. We organized our research work around the three aforementioned axes.

1.2.3 Axis 1: Extraction of Structured Data From Log Files

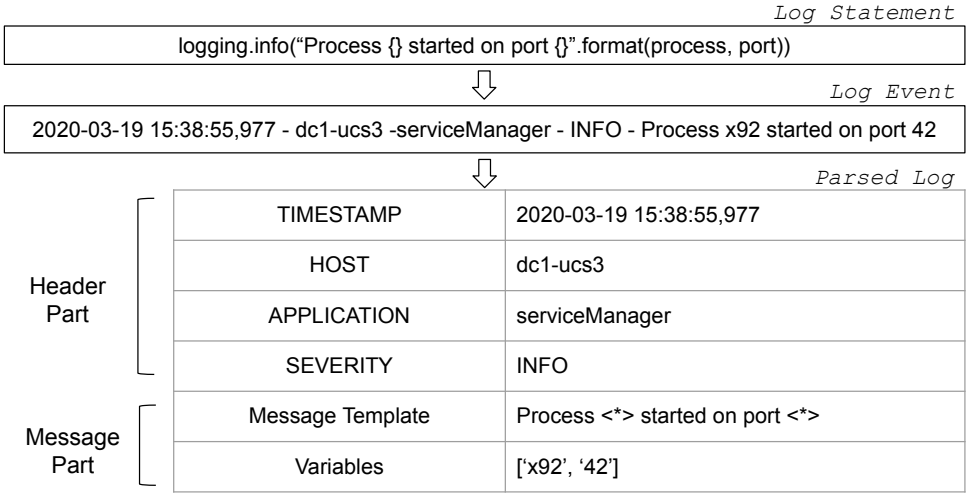


Figure 1.3: Log Statement, Log Event and Parsed Log, an Example

Programs generate log events during runtime, which are then stored in log files. Log events constitute semi-structured data, featuring a header with a consistent format, while the message content is dictated by the developers' discretion. Log messages are generated by injecting variables linked to the real-time state of a system inside a code-embedded message template. Figure 1.3 displays a log statement in Python and an example of a log event generated by this statement.

As discussed, logs are created according to a predefined format (e.g., rsyslog RFC5424 [25]) and can be divided into two parts:

- A header, composed of different predefined fields. In the Figure 1.3 example, the header is composed of 4 distinct fields (*TIMESTAMP*, *HOST*, *APPLICATION*, and *SEVERITY*). As the header formatting is defined as software level, all the log headers will follow the same format.

- A message, which is a text field without format constraints. Due to its flexibility, it is common to embed information describing the current state of the system inside the message.

Mining log-embedded information in contexts where you don't have access to the source code is a common issue for automated log-based applications. Log parsing is an active research topic [1] and, a large panel of approaches have been proposed during the last two decades. To deal with the unstructured format of log messages and provide them with structured inputs, existing log-based anomaly detection methods rely on log parsing algorithms to mine logs underlying templates. As the quality of the parsing process has a direct influence on the precision, and the efficiency of the downstream log-based applications, and to be relevant in a cloud environment, a parsing solution must be able to work online, efficiently, and be robust to the evolution of the log statements.

Alongside this data mining axis, we explore state-of-the-art log parsing solutions and their relevance for a cloud logging environment, as well as ways of exploiting log-embedded information. Particular attention was given to the robustness of the existing solutions and of our

proposals, as well as their ability to process large volumes of logs efficiently.

1.2.4 Axis 2: Anomaly Detection in Highly Distributed IT Infrastructures

Due to their abundance and the information they contain, the logs are one of the most valuable data sources for anomaly detection [26; 27]. Different log-based anomaly approaches have been proposed within the previous years based on traditional machine learning techniques and more recently on neural approaches [28] such as LSTM [24; 29] or Transformer structures [30]. Deep learning techniques employ trained models to anticipate the subsequent element in a log template sequence, flagging an anomaly if the predicted probability falls below a set threshold. Those methods have strong claims regarding their performances, however, in practice, we found that knowing or inferring log relationships to identify log sequences is a challenging and often not possible task within a cloud environment. This is due to logs most of the time not having a unique operation identifier. Also, inferring related events is hard due to the multiplicity of logging devices, the involved volumetry and the temporal proximity of operations. Such constraints limit the application of previously cited anomaly detection methods to the monitoring of specific cloud components (e.g., the API level where you are likely to find request-id).

Alongside this axis, we explore automated anomaly detection systems that meet the requirements of volume, variability and evolution of a cloud platform logs such as 3DS OUTSCALE one.

1.2.5 Axis 3: Contextualization of Anomalies for Decision Support

When an anomaly is detected, the information sent back to a human team must indicate as precisely as possible the cause and the context of the problem. To ensure effective resolution, identifying the context is as important as the anomaly itself. The classification of an anomaly serves, first of all, to target the most qualified experts to manage it. Work-related to this axis includes a literature review of the existing classification systems for log-based anomalies and their relevance in a cloud computing context.

Due to the constant evolution affecting the logging behavior of the underlying components of a cloud platform, classifying anomalies is a tedious and not relevant in the long-term task. Also, professionals may have different ways of handling the same anomalies depending on their context. Those concerns drove us to propose and evaluate a context-aware anomaly detection system that outputs anomalies alongside their context. This context includes the concerned equipment and applications as well as ways of evaluating its criticity. This output allows practitioners to build their own business rules on top of it instead of forcing them inside a classifier.

1.3 An Overview of Our Contributions

To be efficient in a cloud computing environment, a parsing method should be accurate, efficient, and able to work in an online fashion. When working with existing log-based anomaly detection methods, we noticed that poor parsing quality drastically reduces the efficiency and precision of the considered approaches. Our experiments with state-of-the-art online parsing methods reveal that hyperparameter tuning and the use of preprocessing regular expressions have a significant impact on the parsing accuracy. This lack of robustness is a major concern in our cloud computing context as it is impracticable to label logs to correctly tune them. In response to this, we introduced USTEP, a log parsing algorithm. This evolving tree-structured algorithm can discover and encode new parsing rules while processing logs online. USTEP is, to the best of our knowledge, the only parsing method that achieves constant parsing time where other methods slow down with new template discovery. USTEP is robust and does not require any prior knowledge regarding the logging environment. We evaluate our proposal against four state-of-the-art online methods over 14 datasets coming from real-world applications. USTEP demonstrated superior performance in both effectiveness and robustness. We also introduced USTEP-UP, a way of running decentralized USTEP instances to ensure the scalability of the parsing step.

Those contributions to the log parsing field led to four publications:

- Vervaet, Arthur, Raja Chiky, and Mar Callau-Zori. "Ustep: Unfixed search tree for effi-

cient log parsing." 2021 IEEE International Conference on Data Mining (ICDM). IEEE, 2021. [31]

- Vervaet, Arthur, Yousra Chabchoub, Mar Callau-Zori, and Raja Chiky. "Online Log Parsing Using Evolving Research Tree." Knowledge and Information Systems: Accepted, to be published

- Vervaet, Arthur, Raja Chiky, and Mar Callau-Zori. "Automatisation de la structuration des logs pour le cloud computing." Extraction et Gestion des Connaissances: Actes EGC'2021 (2021). [32]

- Vervaet, Arthur, Raja Chiky, and Mar Callau-Zori. "USTEP: Structuration des logs en flux grâce à un arbre de recherche évolutif." Extraction et Gestion des Connaissances: EGC'2022 38 (2022). [33]

To facilitate its use by practitioners and researchers, we have made the source code of USTEP[1] publicly available.

When reviewing log-based anomaly detection state-of-the-art, we noticed that existing work was focused on detecting anomalous log sequences. In practice, the reconstruction of log sequences in a cloud computing context is often impossible, due to a lack of unique identifiers as well as the technical impracticability to link logs from two different sources related to the same event. As we could not apply existing log-based anomaly detection methods due to prerequisite conflicts, we chose to explore new log-based ways of detecting abnormal events within large composite platforms.

The design of Monilog, our automated log-based anomaly detection system, is inspired by the field of multivariate time series. Monilog uses models to forecast the log traffic and the resulting error time series is computed using this forecast and the real traffic to detect anomalous patterns to detect abnormal behavior of equipment. Monilog has the capacity to generate detailed reports, regarding anomaly candidates as well as the implicated system and applications. Monilog doesn't

[1] https://github.com/outscale/ustep-online-log-parser

require any prior knowledge regarding the relationship between logs, and can work in a streaming fashion allowing close to real-time alerting. We implemented and tested our proposal at a cloud scale using all the syslog messages [25] generated by the Kernel Based Virtual Machine of 3DS OUTSCALE European cloud region for 11 consecutive days. Our system successfully predicted all three reported server crashes at least 80 minutes prior to their occurrence. Our precision evaluation also displays the relevance of the other reported abnormal events.

Monilog is the corner stone of this thesis, related work led to two publications:

- Vervaet, Arthur, Yousra Chabchoub, Mar Callau-Zori, and Raja Chiky. "Monilog: Detection of Anomalies in Cloud Computing Infrastructures using Logs" - Under review
- Vervaet, Arthur. "MoniLog: An Automated Log-Based Anomaly Detection System for Cloud Computing Infrastructures." 2021 IEEE 37th International Conference on Data Engineering (ICDE). IEEE, 2021. [34]

1.4 Organization of This Manuscript

The remainder of this manuscript is organized as follows. In Chapter 2, we present existing usages and applications based on logs as well as the state-of-the-art on automated log-based anomaly detection and log parsing. In Chapter 3, we study the impact of parsing accuracy on AD method's accuracy and the impact of tuning on existing methods. Chapter 4 is dedicated to USTEP, our proposal log parsing solution workflow and memory structure, the USTEP-UP architecture, and our perspectives regarding the log-parsing problem. Chapter 5 is dedicated to Monilog, our automated log-based anomaly detection system. We discuss here its architecture and state-of-the-art limitations. In Chapter 6, we present experimental results of Monilog applied to the monitoring of Virtualization servers at cloud scale. Chapter 7 sum-ups the presented work and discuss future work based on this thesis.

CHAPTER 2

State of the Art

Contents

In this Chapter, we present a state-of-the-art of automated log-based anomaly detection (Section 2.3), and we discuss identified limitations of the existing methods in particular for their application to the cloud-computing platform field (Section 2.6). This chapter also includes a state-of-the-art of log parsing domain as well as an overview of some log-mining applications.

2.1 Log Mining Applications Within the Literature

Logs are widely available data sources containing real-time information regarding the states of a system, and they have already proved their utility for a wide range of mining tasks such as user, or resource profiling. With the increasing amount of generated log lines, multiple automated log-based approaches have been proposed [35; 36] to exploit the diversity of information they contain. In this section, we present examples of possible exploitation of the logs that we found in the literature. We do not intend to present an exhaustive list of existing applications, what follows should in no way be considered as a survey but rather as an inspiration and an opening on the existing usages of logs.

2.1.1 Root Cause Analysis

Root cause analysis is one of the most common and traditional uses of logs [37]. By examining logs preceding a system failure or other undesirable state, one can reconstruct the sequence of events leading to the system's failure. One challenge is that log files are typically designed to represent a single stream of events. Messages from multiple sources, however, may be interleaved both at runtime (from multiple threads or processes) and statically (from different modules of a program). For static interleaving, header-embedded information can help filter out logs and focus on the one related to the considered context. However, for runtime interleaving, a thread ID does not solve the problem because a thread can be reused for independent tasks. This is a major challenge when performing root cause analysis on a supercomputer, leading to the

proposal of tools like LogAider [38], to mine potential correlations of HPC log events.

2.1.2 Dissection of Performance Issues

Log analysis can help optimize or debug system performance. Understanding a system's performance is often related to understanding how the resources in that system are used. Distributed systems are notoriously difficult to get right, and we came across several research papers proposing log-mining approaches to dissect and improve the performances of such systems. Wang et al. [39] proposed a log-based approach to detect the root causes of writing data on parallel file performance bottlenecks. By analyzing logs collected on Cray XC40 supercomputer system used for scientific purposes, authors produced guidelines to help developers optimize the I/O behavior of their applications. Tan et al. [40] explored a log-based approach for performance debugging of MapReduce, a programming paradigm and framework [41] for parallel distributed computations on commodity clusters. Lu et al. [42] focused their work on detecting concurrency bugs in distributed systems by the generated logs. Concurrency bugs are triggered by complex interleaving of messages and are difficult for programmers to correctly reason about and handle concurrent executions on multiple machines. The use of the proposed log mining approach alleviates the pain of hands-on root cause analysis in such scenarios.

2.1.3 Detection of Intrusions and Attacks

Logs are commonly used for security purposes. Searching log files can help detect intrusion traces or identify malicious attack patterns. When searching for attacks, pattern matching is an efficient way of looking for traces of classical malicious behaviors. With this approach, only known patterns can be recognized, yet new types of attack may appear with only small changes made to existing patterns. This explains why more recent approaches are focused on machine learning techniques to detect anomalous behaviors to be more robust to attacker's small changes. As logs are a broadly available data source in many systems, they are used by a wide range of security applications. We came across methods using logs to detect SQL injections, a classic

attack to mine or alter the content of a database [43]; or detect malicious behaviors using proxy logs [44]. There are also examples of log-mining applications that reconstruct behavior graphs to detect intrusions or other types of unusual behaviors [45; 46; 47].

2.1.4 Failure Prediction

Failure prediction also forms a dynamic focus area within the log mining field. Logs have been used by Gao et al. [48] and Chaves et al. [49] to predict hard disk drive failures. Considering the varied brands and models of drives with different input/output workload patterns present in data centers, their approach becomes particularly pertinent [50]. Log-based failure prediction has also been successfully applied for detecting failures in HPC nodes [51] and Oracle databases [52].

Failure prediction is close to anomaly detection; however, failure prediction methods are designed for one system, or one class of system and usually exploit context-specific information to perform their task, whereas anomaly detection methods aim to be system agnostic. The remainder of this manuscript is focused on log-based anomaly detection methods.

2.2 Anomaly Detection

2.2.1 Example of Applied Research Work for Different Fields

Due to its practical utility, anomaly detection has emerged as a significant topic area across various fields of application [53]. In the medical domain, there are several recent examples [54] of work on the detection of tumors [55; 56] or, brain lesions [57; 58] using deep learning techniques. Same for the banking sector, where anomaly detection techniques are for instance applied to fraud detection in the use of credit cards based on debit records [59; 60]. Closer to our context, we retrieve work on network intrusion detection systems [61; 62].

2.2.2 Within the Cloud Computing Domain

The development of large-scale services and their underlying systems typically involves the collaboration of hundreds of contributors, divided across different teams and sometimes, distinct

organizational structures. Developers and operators usually have incomplete information regarding the overall system and tend to determine anomalous events from a local perspective, which is error-prone. Besides, manually detecting anomalous log sequences is not efficient nor scalable because of the continuous increase in the volume of logs generated by modern systems.

The use of keyword-matching techniques or regular expressions helps to detect simple and well-known anomalous events by seeking characteristic patterns. Although such pattern-matching approaches are unable to identify a large portion of the anomalies, as many of them are sequences of *"non-anomalous"* logs leading to an undesired outcome. Moreover, rule-based approaches are sensitive to changes in the code base. This need for automation and robustness are reflected in numerous research works on automated log-based anomaly detection methods [63].

2.3 Automated Log-Based Anomaly Detection

Year	Paper	Method	Year	Paper	Method
2007	[64]	SVM	2019	[29]	Bi-LSTM
2009	[65]	PCA	2020	[66]	IF + Autocencoder
2012	[67]	IM	2020	[68]	BERT + Bi-LSTM
2016	[69]	Clustering	2021	[70]	Transformer
2017	[71]	LSTM	2021	[72]	Clustering + CNN
2017	[73]	Clustering	2021	[74]	Transformer
2018	[75]	CNN	2021	[76]	GAN
2018	[77]	Clustering	2021	[78]	Adversarial Network
2019	[79]	Transformer	2021	[80]	Transformer
2019	[24]	Bi-LSTM	2022	[81]	TCN

Table 2.1: Log-Based Anomaly Detection Methods, an Overview

During the last two decades, automated log-based anomaly detection has been an active research topic, and a wide range of methods have been proposed based on different techniques [36; 82; 83]. Since 2016, the rise of the deep learning field led to the apparition of several methods based on neural network structures outperforming traditional machine learning-based methods. According to Bhanage et al. [36], over 87 different publications related to log-based anomaly detection have been issued between 2016 and 2021. In this section, we cover only the most

representative methods and their specificities. Table 2.1 presents a summary by years of the selected methods alongside the original papers and the used techniques.

2.3.1 Traditional Methods

To the best of our knowledge, the first reported log-based anomaly detection method was brought up by Liang et al. in 2007 [64]. The proposed system is based on the traditional Support Vector Machines (SVM) set of classifiers and aims to detect failures affecting IBM BlueGene/L supercomputer.

In 2009, Xu et al. proposed a method based on the Principal Component Analysis (PCA) algorithm [65] to detect system runtime problems. For their evaluation, they used log traces generated by Hadoop[1], an open-source software dedicated to the management of big data files. They reconstruct the log sequences using code-embedded log statements for perfect parsing and display the relevance of PCA for detecting anomalous log sequences in an online fashion.

Lou et al. published a method based on the Invariant Mining (IM) technique [67] to mine program workflow using logs. The authors artificially generated log traces using Hadoop and JBoss[2], two open-source programs. The original work is focused on inferring log relationships and reconstructing log sequences, but their work can be extended as an anomaly detection method by considering unusual workflow as anomalous sequences, that's why we chose to mention this method here.

Multiple methods based on clustering techniques exist within the literature, such as Log cluster [69] proposed by researchers from Microsoft to identify problems within two internal online services. Wurzenberger et al. [73] also used a clustering-based approach to detect cybersecurity threats (e.g., SQL-dump, SQL-injection, brute force login attack) within a workstation running a virtual server for an Apache Web server[3] hosting a MANTIS Bug Tracker System[4], a MySQL database[5] and a reverse proxy.

[1] https://hadoop.apache.org/
[2] https://www.redhat.com/fr/technologies/jboss-middleware/application-platform
[3] https://httpd.apache.org/
[4] https://www.mantisbt.org/
[5] https://www.mysql.com/

2.3.2 Deep-Learning Based Methods

In 2017, Du et al. proposed Deeplog [71], a log-based anomaly detection method based on an LSTM neural network. Deeplog model logs as natural language sequences and can process them in a streaming manner. According to an experimental evaluation conducted by Deeplog authors, it achieves higher accuracy than the previously presented non-traditional methods. Deeplog is the first identified deep-learning based method and it led the way to multiple approaches driven by the progress in this domain.

Deeplog approach was directly extended by LogRobust [24], LogAnomaly [29] and SwissLog [68], three methods based on bidirectional LSTM neural networks aim to detect sequential log anomalies. Authors of each method identified log statement instability as a significant issue, proposing innovative embedding mechanisms to mitigate the effects of log concept drift on their models. To regroup new templates with an existing one to keep unchanged the embedding vector size, LogRobust uses semantic information of the log event, LogAnomaly generates an intermediate template using Ft-tree [84], and SwissLog uses a BERT [85] encoder to encode log templates.

Convolutional Neural Network (CNN) has been proposed to capture local semantic information instead of global information and defeat the overfitting issues in regular neural networks. It has become one of the most representative neural networks in the field of deep learning [86]. This structure has allowed significant advances in the field of computer vision, but it has also been successfully applied to time series prediction and signal identification problems. Closer to our context, CNN has been used by Ren et al. to assign a category (e.g., Network, Memory, Security, DataBase ...) to log events [87] based on their message. In 2018, Lu et al. [75] proposed a CNN-based approach able to learn event relationships in system logs and detect anomalies based on this information. More recently, Yang et al. introduced PleLog [72], a log-based anomaly detection approach that combines a clustering step and a Convolutional Neural Network. The clustering step is used to assign a label to log events and allows the system to train more efficiently within unsupervised contexts.

The great success of the Bidirectional Encoder Representations from Transformers (BERT) in modeling sequential data [85] inspired multiple Transformer-based approaches. In our context, we have identified two different types of BERT usages: 1/ Nedelkoski et al. [79], and Guo et al. [74] train such model to capture patterns of normal log sequences and used it later on to detect anomalous ones; 2/ Previously depicted method SwissLog uses only the encoder part of Bert to enhance the robustness of the addressed approach to new logging patterns.

Finally, we want to highlight here four log-based anomaly detection methods based on different techniques that we found interesting: 1/ Farzad et al. [66] used two deep autoencoder networks to extract features from logs and perform anomaly detection using the Isolation Forest (IF) algorithm [88]; 2/ LogGAN, proposed by Xia et al. [76] an LSTM-based generative adversarial network for anomaly detection; 3/ QLLog is another adversarial method based on the Q-learning algorithm, a reinforcement learning method based on value function; 4/ LightLog, a light anomaly detection method to handle large-scale logs on edge devices with limited computed power. Lightlog uses a modified Temporal Convolutional Network (TCN) to detect anomalies.

2.4 An Introduction to Log Parsing

Log parsing is a classic first step for log-based anomaly detection and log-based applications in general. Log parsers aim to extract the underlying information of a log event without assessing the associated log statement. In practice parsing the header is straightforward as it follows a given format, however, separating the injected variables from the message template is a challenging task. The application of regular expressions is a straightforward method to parse raw log messages; thus, this requires listing all the logging statements and generating the appropriate expressions. In practice, the source code is not always accessible, and keeping track of all the changes is a tedious task [89]. Log-parsing methods aim to tackle this issue by providing an automated way of mining log templates from log messages. Previously presented Figure 1.3 illustrates the expected parsed version of a sample log line.

2.4.1 Problem Formulation

In this manuscript, we will adopt the formalism introduced by *Nedelkoski et al.* [30] regarding the log-parsing problem. Logs are defined as sequences of temporally ordered unstructured text messages $\mathcal{L} = (l_i : 1, 2, ...)$, with i the positional index of a log message l_i within the sequence. Tokens are the smallest inseparable singleton objects within a log message. Each log message is constituted of a finite sequence of tokens (words) separated by spaces $\mathbf{t_i} = (t_j^i : t \in \mathcal{T}, j = 1, 2, ..., |\mathbf{t_i}|)$. With $\mathcal{T}$ the set of all tokens, j the positional index of a token within a log message l_i, and $|\mathbf{t_i}|$ the total number of tokens inside l_i. Tokenization is defined as a transformation function $\mathcal{M} : l_i \rightarrow t_i, \forall i$.

Log parsing aims to structure log messages by separating their constant (template) part, from the variable ones. A log parsing method is defined as a function $f : t_i \rightarrow (e_i, v_i)$, with e_i a template, and v_i a list of variables.

2.5 Log Parsing State of the Art

During the last two decades, different approaches to the log parsing problem have been proposed [1; 35; 90]. According to the usage workflow, log parsers can be classified in two approaches: Offline algorithms process logs in a batch manner updating the parser from time to time, this updated parser is used in production to structure incoming logs; Online algorithms process logs one by one in a streaming fashion and update the parser on the fly. In the following, we discuss the method, the opportunities, and the limitations of state-of-the-art parsing solutions.

2.5.1 Offline parsing

In surveys, we find algorithms based on different techniques such as *clustering* (LKE [91], LogSig [92] and LogMine [93]), *iterative partitioning* (IPLoM [94]), *frequent pattern mining* (SLCT [95], LFA [96] and LogCluster [97]), or *heuristics* (AEL [98] and Logram [99]). All those log parsing approaches perform multiple passes on a batch of log messages making this process

is costly both in terms of memory and compute utilization.

Recently, in a context closely related to ours, *Nedelkoski et al.* [30] proposed NuLog, a log parser based on the transformer architecture [85]. Their method displays strong parsing accuracy on datasets coming from different applications. They also provide two case studies illustrating the relevance of log parsing to feed anomaly detection models. However, training their model requires a labeled dataset representative of the system logs statements. In practice, such a dataset is hard to obtain, and as observed by *Zhang et al.*, log statements evolve. To remain relevant a Nulog-based log parsing system would need to be trained regularly with newly labeled data.

2.5.2 Online parsing

Online parsing methods handle logs in a streaming manner, enabling the processing of large datasets by reducing memory bottlenecks. Even if it can help them tune their parameters, such approaches do not require labeled data to train. To the best of our knowledge, 4 online log parsers have been proposed within the literature: SHISO [100] and LenMa [101] which are based on clustering techniques; Spell [102] which uses the longest common subsequence algorithm to match templates; and Drain [103; 104] a log parsing approach based on a fixed depth research tree. Those methods can detect new templates on the run, and adapt their parsing to new log statements. Among them, Drain is often described as the most effective method and is cited as the recommended log parser by different studies on log-based anomaly detection [1; 29; 105].

Online parsing methods all use a memory structure to remember the intrinsic characteristics of the previously parsed logs: Spell creates and maintains an LCSmap with the different templates it found; Drain encodes new parsing rules inside a research tree; SHISO and LenMa maintain clusters of log templates. This memory structure plays an important role in the space and time complexity, and therefore the efficiency of each method (Table 4.6).

2.5.3 Distributed parsing

Parsing time can be a bottleneck for downstream applications. Batch parsing methods are easily distributable (e.g., Logram [99]), as their knowledge base doesn't evolve after the training phase. However, they can't adapt to new log statements, and with the exception of NuLog are less effective than the best online methods according to Zhu & al. benchmark [1]. Solutions were built on the top of spark for quick parallelization. He et al. [106] proposed a parallel log parsing method (POP) using some heuristic rules and hierarchical clustering. Zhu et al. [1] ran an extension of Drain in production for more than one year with a 90% accuracy. In [107], authors propose Logan based on the longest common subsequence approach that parallelizes the training phase.

2.5.4 Parsing Accuracy Metric

Parsing Accuracy (PA) is a commonly used metric to quantify the effectiveness of automated log parsing solutions. It is defined by [102], and [1] as the ratio of correctly parsed log messages over the total number of log messages. Parsing operations associates each log message with a template and a set of variables. A log message is considered correctly parsed if and only if its event template corresponds to the same group of log messages as the ground truth does. For example, if a log sequence [E1, E2, E2] is parsed to [E1, E2, E3], we get PA=1/3, since the 2nd and 3rd messages are not grouped. In contrast to standard evaluation metrics that are used in previous studies, such as precision, recall, and F1-measure [94; 108; 109], PA is a more rigorous metric, as with PA, partially matched events are considered incorrect.

2.6 Discussion

As outlined in the previous sections, log-based anomaly detection is an active research topic. Recent methods are mainly based on deep learning techniques and follow the trend of advances in this field. Despite the variety of proposed approaches, all the presented methods follow a common workflow. First, they use a log parsing algorithm or an encoder structure to mine logs

underlying information. Then they use a previously trained model to predict the next element of a given log template sequence. An anomaly is flagged if the predicted probability of the verified next log template of a given sequence falls below a specified threshold.

2.6.1 Reconstructing Log Sequences

When trying to work with existing methods in the 3DS OUTSCALE cloud computing context, we run into several shortcomings. First, we found that reconstructing log sequences is not always possible in a cloud computing environment due to the volumetry and the different layers of components.

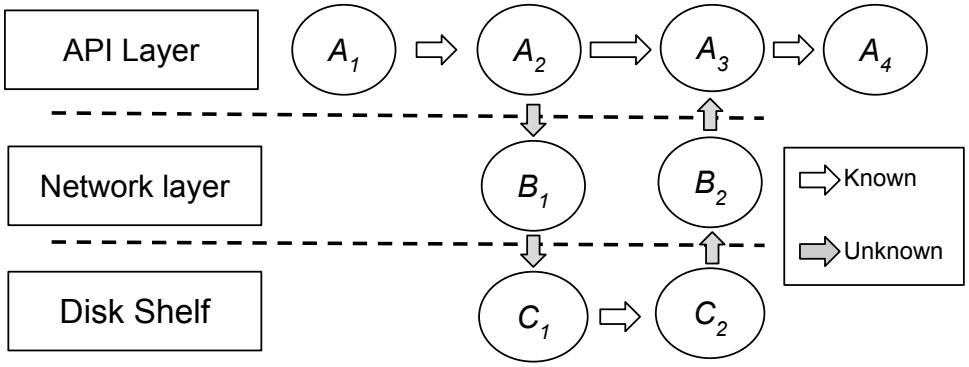


Figure 2.1: Multi-Layer Logging Architecture

Figure 2.1 describes a logging environment with 3 layers: an API layer where we can correlate logs using an identifier (e.g., *request_id*); a network layer where we cannot have any relationship information as it can englobe multiple routers and no tracking identifier is logged; a disk shelf layer where we can link related logs using an identifier. Gray's arrows represent effective but not discoverable relationships from a logging point of view. In this context, we are not able to retrieve and order the full log sequence related to a given event such as a user asking for a specific virtual machine to be deployed.

This limitation was our main motivation for the design of Monilog, a log-based anomaly detection system capable of functioning within complex environments such as a cloud-computing platform. We wanted Monilog to operate without any knowledge about the relationship between

logs making it a suitable solution for multi-source logging environments. Thanks to our access to 3DS OUTSCALE, we were able to apply our proposal to the monitoring of a real-life cloud computing platform. More details regarding this system are to be found under Chapter 5 that presents its architecture and Chapter 6 that reports experimental results regarding its ability to forecast anomalous events within components of the 3DS OUTSCALE cloud platform.

2.6.2 A Limited Number of Open Datasets

Except some internal systems from Microsoft, methods in the literature are evaluated on a limited pool of log datasets. We think this is due to the need for reproducibility and easy comparison with the existing methods but also to the limited number of datasets available open-source. More specifically, four datasets containing labeled anomalies are frequently used to compare approaches [110]:

- BGL [111], a collection of logs issued from a BlueGene/L supercomputer system.
- Hadoop Distributed File System (HDFS)[1], is a distributed file system that aims to provide high-throughput access to application data. This log set [112] was generated in a private cloud environment using benchmark workloads and manually labeled through handcrafted rules to identify the anomalies.
- Hadoop[2] is a big data processing framework that allows for the distributed processing of large datasets across clusters of computers using simple programming models. The associated dataset [69] was collected from a Hadoop cluster with 46 cores across five machines running two different test applications. Failures were manually injected into parts of the dataset to simulate machine down, network disconnection or disk full issues.
- OpenStack[3] is a cloud operating system that controls large pools of computing, storage, and networking resources throughout a data center. This dataset [71] was generated on

[1]https://hadoop.apache.org/
[2]https://hadoop.apache.org/
[3]https://www.openstack.org/

CloudLab, a flexible, scientific infrastructure for research on cloud computing.

The sensitivity of information contained in logs from real environments often makes their publication infeasible, moreover, labeling anomalies in such datasets is a tedious and time-consuming task. We believe this explains why such a small pool of open-source log traces has been studied through the years. One of the drawbacks of this phenomenon is that it becomes complicated to evaluate the relevance of the proposed methods in more complex log environments. Most of the previously presented methods achieve a 0,94+ precision on each of the considered datasets, therefore it can be hard to quantify the contribution of a method compared to the existing ones.

Anomalies affecting cloud computing platforms can be way more complex than the one listed in the open-source datasets. They can affect a vast range of interconnected components and therefore may require a multi-log traces approach to be detected [34]. Making a log dataset available for study is a challenging task as log data provides all the details about the execution of the infrastructure components, and the misuse of this data may cause serious problems. Due to confidentiality issues and strict security policies, it is not possible to open-source logs coming from 3DS OUTSCALE internal services. However, we paid attention to providing the maximum of information regarding the complexity of its logging environment and the lesson learned. We hope our results will help to understand the challenges raised by the monitoring of large online systems such as the cloud computing platforms and the complexity of their logging environment.

CHAPTER 3

Online Log Parsing Methods and Hyperparameters Tuning

Contents

3.1 Log Parsing for Log-Based AD Methods

Log parsing plays a crucial role in extracting valuable information embedded within logs and is an essential step for enabling most existing log-based AD methods. Over the years, there have been numerous methods proposed in this active research area, as documented by Zhu et al. [1] in their benchmark. They conducted an evaluation of 13 log parsers across 16 datasets, and Drain emerged with the highest overall performance, reaching an average PA of 0.865. However, it is important to note that the performance of parsing methods can vary greatly across different datasets. For example, Drain achieved a PA of 1.0 on the HDFS dataset, but a significantly lower PA of 0.527 on the Proxifier dataset.

In this section, we focus on investigating the impact of PA on log-based AD methods. We also explore the influence of hyperparameters and preprocessing techniques on PA.

3.1.1 Impact of the Parsing Accuracy on Anomaly Detection Methods

In order to assess the impact of PA on the accuracy of downstream AD methods, we conducted an evaluation using an instance of Deeplog [71]. The evaluation involved utilizing the same log dataset, but parsed with three different PAs. For this purpose, we employed a Drain instance as the parsing solution and deliberately varied the PA by adjusting hyperparameter values.

The dataset selected for this evaluation consists of HDFS system traces [110]. This particular dataset was chosen because it is commonly used by both the authors of Drain [103] and Deeplog in their respective research papers. By applying different PA levels to the parsed logs and subsequently evaluating the performance of Deeplog, we aimed to examine how variations in parsing accuracy may impact the overall accuracy of the downstream AD method.

To investigate the relationship between parsing quality levels and the accuracy of downstream anomaly detection methods, we conducted training on 10 separate instances of Deeplog for each parsing quality level (1.0, 0.8, 0.65). Figure 3.1 showcases the distribution of AD accuracy obtained on the test set by each instance. During our experimentation, we observed that a 20% decrease in parsing accuracy resulted in a significant 75% reduction in AD accuracy. This

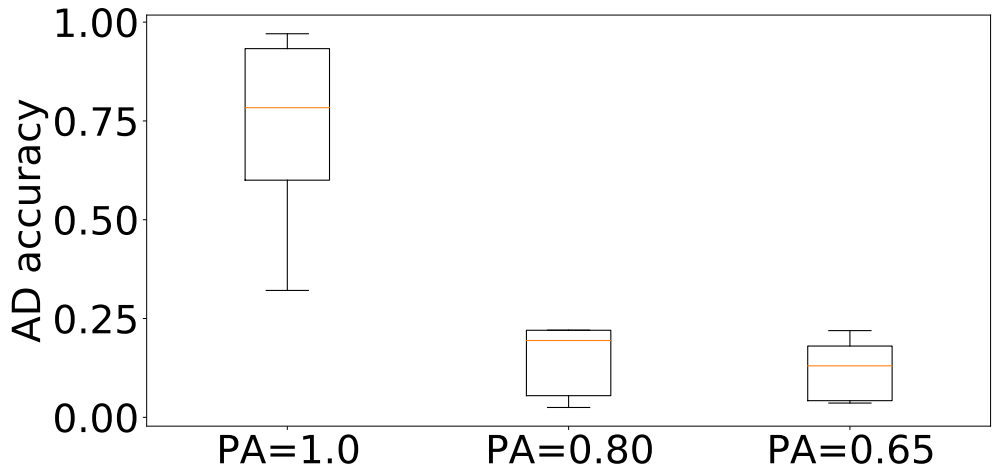


Figure 3.1: Parsing Accuracy Impact on Deeplog Performances

highlights the critical impact of parsing quality on the overall performance of AD methods.

In a study conducted by Zhu et al., it was found that the Drain parsing accuracy fell below 0.8 for 6 out of the 16 considered datasets. Furthermore, other online parsers examined in the study, such as Spell (9 out of 16) and LenMa (7 out of 16 with a PA as low as 0.174 on the HealthApp Dataset), also demonstrated lower PA values for a significant proportion of the datasets. Additionally, a more recent empirical study by Fu et al. [113] further confirms the impact of log parsing on the performance of log-based anomaly detection methods. They highlight that log-based AD methods tend to exhibit poorer performance on datasets with higher numbers of underlying log templates, indicating the challenge posed by diverse datasets with a large number of underlying templates.

3.1.2 Parsing Errors

The results obtained from our experiments highlight the significant influence of log parser precision and accuracy on the overall performance of the parser/anomaly detection model system. The presence of parsing errors introduces new log templates, which in turn artificially increases the input vector of deep learning models, resulting in reduced learning quality. Furthermore,

we observed that the time complexities of existing online log parsing approaches are directly affected by the number of templates discovered. In systems with diverse logging structures, the parser's processing speed can be noticeably slower. This aspect becomes particularly critical for systems that require real-time processing, such as AD systems operating in complex logging environments.

In our specific case, where efficiency is a crucial factor in achieving real-time processing, log parsing efficiency becomes a sensitive issue. The ability to process logs efficiently and in a timely manner is of utmost importance for AD systems operating in complex logging environments.

3.2 Online Log Parsers and Hyperparameters Tuning[1]

Log parser	Year	Technique	#hyperparameters
SHISO	2013	Clustering	4
LenMa	2016	Clustering	1
Spell	2016	LCS	1
Drain	2017	Parsing tree	2

Table 3.1: Online Log Parsers Presented in [1].

Two notable studies conducted by He et al. [108] and Zhu et al. [1] have focused on evaluating the accuracy and robustness of log parsing solutions using log traces from various applications. These studies provide valuable insights into the potential applications of existing methods for online log processing. The evaluated methods commonly employ a preprocessing step to filter out certain tokens and can be fine-tuned using various hyperparameters (as depicted in Table 3.1).

In this section, our focus shifts towards examining the influence of these two factors, the preprocessing step and the selection of hyperparameters—on the PA of two state-of-the-art log parsing solutions: Spell and Drain.

[1]The work presented in this section is mainly extracted from our paper Vervaet, Arthur, Raja Chiky, and Mar Callau-Zori. "Automatisation de la structuration des logs pour le cloud computing." Extraction et Gestion des Connaissances: Actes EGC'2021 (2021). [32].

3.2.1 Preprocessing

Preprocessing is a well-established initial step in log parsing processes. Its purpose is to filter out specific tokens based on user-defined regular expressions (regexes) that represent commonly encountered variables, such as IP addresses, file paths, or URLs.

During our analysis of logs from a 3DS OUTSCALE internal service, we made an interesting observation. A pproximately 60% of the tokens within log messages originated from JSON or XML-formatted data. In API-like services, it is common practice to append such formatted data at the end of a log line, as it provides valuable context to understand the log entry (e.g., "Send 42 bytes to 121.13.4.26 {user_id=125, service_name=dart_vader}"). Given that JSON and XML are already structured formats, it becomes advantageous to exclude these parts from the parsing process. By doing so, we can reduce the average length of log messages and enhance parsing efficiency and accuracy.

3.2.2 Experimental Motivations

Given the potential impact of preprocessing and parameter choices on the precision and efficiency of log parsing approaches, the selection of appropriate values becomes crucial. However, in many industrial contexts, evaluating the parsing accuracy can be challenging due to difficulties in labeling the data. Factors such as the evolution of log statements and limited access to source code make it complex to determine the optimal parameter values. In our study, we specifically address the following research questions:

- ***RQ1.*** *What is the impact of preprocessing and hyperparameter tuning on the precision and accuracy of log parsing?*
- ***RQ2.*** *Is it possible to identify generic hyperparameter values that ensure a good parsing accuracy across different log datasets?*

By addressing these research questions, we strive to provide insights and guidance for practitioners in selecting optimal preprocessing techniques and hyperparameter values, ultimately

enhancing the precision and efficiency of log parsing approaches.

3.2.3 Experimental Context

Dataset	Size	#Messages	#Templates	#Unique Tokens
OpenStack	60,01 MB	207 820	51	942
Android	3,38 GB	30 348 042	76 923	3 599
HDFS	1,47 GB	11 175 629	30	1 445

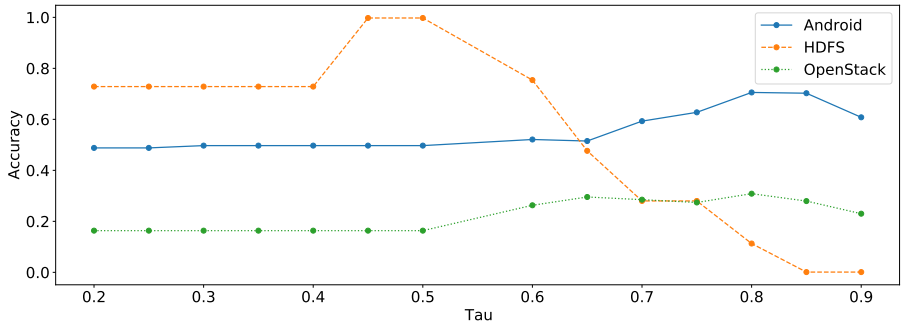
Table 3.2: Characteristics of Datasets Used by [1] Benchmark

For our study, we have chosen to evaluate the performance of Drain and Spell, which are considered the top-performing online log parsing solutions according to the benchmarks conducted by Zhu et al. [1]. and He et al. [63]. To assess their performance, we have utilized a dataset consisting of 2 000 labeled log events from three different systems: Android, HDFS, and OpenStack (as shown in Table 3.2).

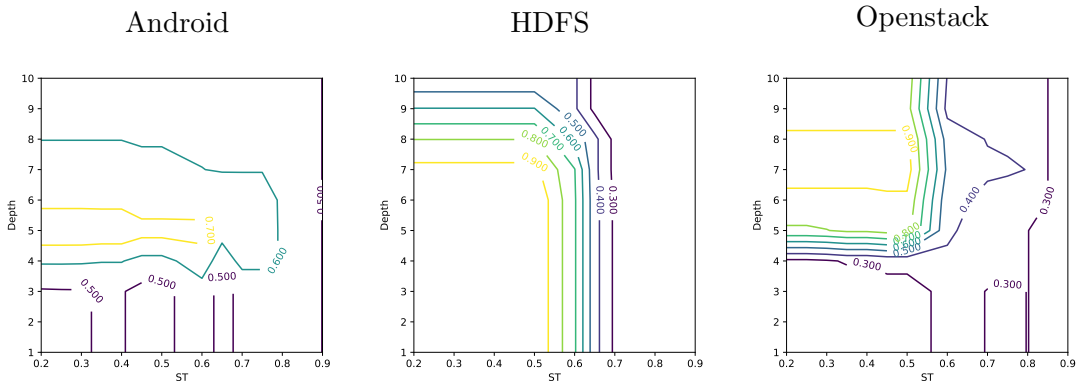
OpenStack is a cloud solution that enables the deployment of Infrastructure as a Service (IaaS) platforms. The Android dataset represents logs from the popular mobile operating system developed by Google, encompassing various patterns and logs related to resource allocation, process management, network management, and more. The Hadoop File System (HDFS) dataset, widely used in the literature, serves as a practical baseline due to its simplicity, featuring a lower number of patterns and variable parts. For further details regarding these open-source datasets and their collection methods, we refer to the paper by He et al. [110], which provides comprehensive information. By evaluating Drain and Spell on these diverse datasets from different logging environments, we aim to gain insights into the performance and effectiveness of these log parsing solutions in real-world scenarios.

3.2.4 Impact of the Hyperparameters on the Parsing Accuracy

Here, we explore the impact of the values of the hyperparameters on the Parsing Accuracy (PA) of Spell and Drain to know if generic values are possible (*RQ1.*).



(a) Spell



(b) Drain

Figure 3.2: Influence of the Hyperparameters on the Parsing Accuracy

In the context of log parsing, it is important to differentiate between model parameters and hyperparameters. Model parameters are learned by the system during training, such as the weights of a neural network, while hyperparameters are set at the initialization stage and affect the learning process or behavior of the algorithm. Examples of hyperparameters include the number of hidden layers or the number of neurons in each hidden layer [114].

For Spell, there is a single hyperparameter, $\tau \in [0, 1]$, which serves as a threshold to determine whether a log belongs to an existing known pattern. Drain, on the other hand, has two hyperparameters: the depth of its search tree ($depth \in \mathbb{N}^*$) and a threshold ($ST \in [0, 1]$), which determines if a log belongs to an existing group.

The impact of hyperparameter values on the parsing accuracy is illustrated in Figure 3.2. The optimal value of τ that maximizes Spell's accuracy varies across datasets: 0.8 for OpenStack,

Dataset	Regular expressions

OpenStack	`((\d+\.){3}\d+,?)+`
	`/.+?\s`
	`\d+`

Android	`(/[\w-]+)+', r'([\w-]+\.){2,}[\w-]+` `\b(\-?\+?\d+)\b|\b0[Xx][a-fA-F\d]+\b|\b[a-fA-F\d]{4,}\b`
HDFS	`blk_-?\d+` `(\d+\.){3}\d+(:\d+)?`

Table 3.3: Regular Expressions Used in the Preprocessing Step

Dataset	Spell (without)		Spell (with)		Drain (without)		Drain (with)	
	PA	#Templ.	PA	#Templ.	PA	#Templ.	PA.	#Templ.
Android	0.60	425	0.91 (×1.5)	180 (×0.42)	0.67	217	0.91 (×1.4)	171 (×0.79)
HDFS	0.28	684	1.00 (×3.6)	14 (×0.02)	1.00	17	1.00 (×1)	16 (×0.94)
O.Stack	0.23	692	0.77 (×3.3)	451 (×0.65)	0.84	75	0.73 (×0.8)	299 (×3.99)

Table 3.4: Influence of the Preprocessing on the Parsing Accuracy

0.85 for Android, a nd 0.5 for HDFS. The choice of τ has the most significant effect on the HDFS dataset, with half of the considered values leading to precision lower than 0.7, and even dropping to 0. On the other hand, for OpenStack and Android, selecting an appropriate value of τ can result in accuracy improvements of up to 15% and 20%, respectively.

Drain's behavior differs among the datasets. While high values of ST (> 0.7) do not yield good results, the choice of the depth hyperparameter is crucial for obtaining accurate results in OpenStack and Android. In the case of Android, the maximum precision achieved is 0.75, while for the other two datasets it reaches 0.9. The optimal depth value varies: 5 for Android, 6 for HDFS, and 7 for OpenStack. It is evident that the accuracy of both solutions is influenced by the choice of hyperparameter values, and finding an appropriate combination of values for a given context is a non-trivial task.

3.2.5 Impact of the Preprocessing on the Parsing Accuracy

Table 3.4 presents the results obtained for each dataset and method, with and without preprocessing. We evaluated the parsing accuracy and also considered the number of patterns to detect

any potential overclassification of logs. The hyperparameter values used were the ones that resulted in the highest PA with preprocessing. The regular expressions used for this experiment were those proposed by Zhu et al. in their benchmark [1].

For Spell, the PA is significantly increased by a factor of at least 1.5 when preprocessing is applied. The improvement is even more pronounced for the HDFS dataset, with a factor of 3.6. The number of templates also benefits from the preprocessing step, bringing it closer to the actual number: 166 for the Android dataset, 14 for the HDFS dataset, and 43 for the OpenStack dataset.

For Drain, the results vary depending on the dataset. While there is an improvement in PA for the Android dataset, the PA and the number of patterns worsen for the OpenStack dataset when preprocessing is applied. This drop in PA can be attributed to the presence of tokens in the shape of "*b*eb-*d*a-*aa*-ba*b-*d*f*f*c" in three templates. Without preprocessing, these tokens affect one level of the partitioning tree. However, with preprocessing, these tokens are split into several parts, affecting multiple levels of the tree and resulting in a PA of 0 for any log containing this type of word. Spell is unaffected because its accuracy on words following this pattern is already zero.

3.3 Discussion

Based on our study, we can conclude that the selection of hyperparameter values and the implementation of preprocessing regexes have a significant impact on the Parsing Accuracy of log parsers. We have also highlighted the influence of PA on the Accuracy of AD methods, as a low PA can significantly degrade the AD potential of downstream models. It is crucial to find optimal hyperparameter values and appropriate regexes to improve the performance of log parsing.

However, finding the best hyperparameter values and regexes can be challenging, especially in complex logging environments with a high volume of data and multiple applications involved. In the case of the 3DS OUTSCALE logging infrastructure, it is not feasible to label logs with

their templates manually, which adds to the difficulty of fine-tuning existing parsing solutions. Based on the obtained results, we are motivated to explore and propose a more robust log parsing solution in the next chapter. This solution aims to address the challenges faced in enabling log-based anomaly detection methods in a cloud computing context.

CHAPTER 4

USTEP a Scalable and Robust Log Parsing Approach [1]

Contents

[1] The work presented in this chapter is heavily based on our paper Vervaet, Arthur, Raja Chiky, and Mar Callau-Zori. "Ustep: Unfixed search tree for efficient log parsing." 2021 IEEE International Conference on Data Mining (ICDM). IEEE, 2021. [31]

4.1 USTEP: Unfixed Search Tree for Efficient Parsing

4.1.1 Motivations

This section provides a comprehensive overview of our online log parsing method called USTEP (Unfixed Search Tree for Efficient Parsing) [31; 33]. USTEP was specifically designed for log parsing in large-scale traces generated by cloud infrastructure, with a focus on low latency and high robustness. It is worth noting that USTEP is currently the only parsing method known to achieve constant time complexity.

In our comparative evaluation (detailed in Section 4.2), we compared USTEP with four state-of-the-art online parsing methods using 10 datasets derived from real-world applications. The results of our evaluation demonstrate that USTEP outperforms the other methods by improving parsing accuracy by 3% and enhancing robustness by reducing the quartile inter-distance by half when compared to Drain.

The development of USTEP was motivated by the identified shortcomings in terms of robustness and efficiency of existing parsing methods, as discussed in the previous chapter. By addressing these limitations, USTEP aims to provide a more reliable and efficient log parsing solution for handling the complexities of cloud infrastructure log traces.

4.1.2 Detailed Workflow

USTEP employs an evolving tree structure to discover and encode parsing rules while performing online log parsing. A USTEP instance is defined as $\mathcal{U} = \{\mathcal{P}, \sigma, \phi\}$, where $\mathcal{P} = \{\mathcal{V}, \widetilde{\mathsf{E}}, \mathcal{E}\}$ represents the parsing tree, $\sigma \in [0; 1]$ is the similarity threshold that determines when a log is considered to match a template, and $\phi \in \mathbb{N}^*$ is the maximum number of templates that can be associated with a leaf node. $\mathcal{V} = \{v_k\}_{k=1}^{\mathsf{N}}$ is the set of nodes, $\widetilde{\mathsf{E}}$ the set of discovered templates, and $\mathcal{E} \subset \mathcal{V} \times \widetilde{\mathsf{E}}$ the set of node-template links.

4.1.2.1 Evolving Research Tree Structure

The USTEP parsing tree is constructed using four types of elements, as depicted in Figure 4.1:

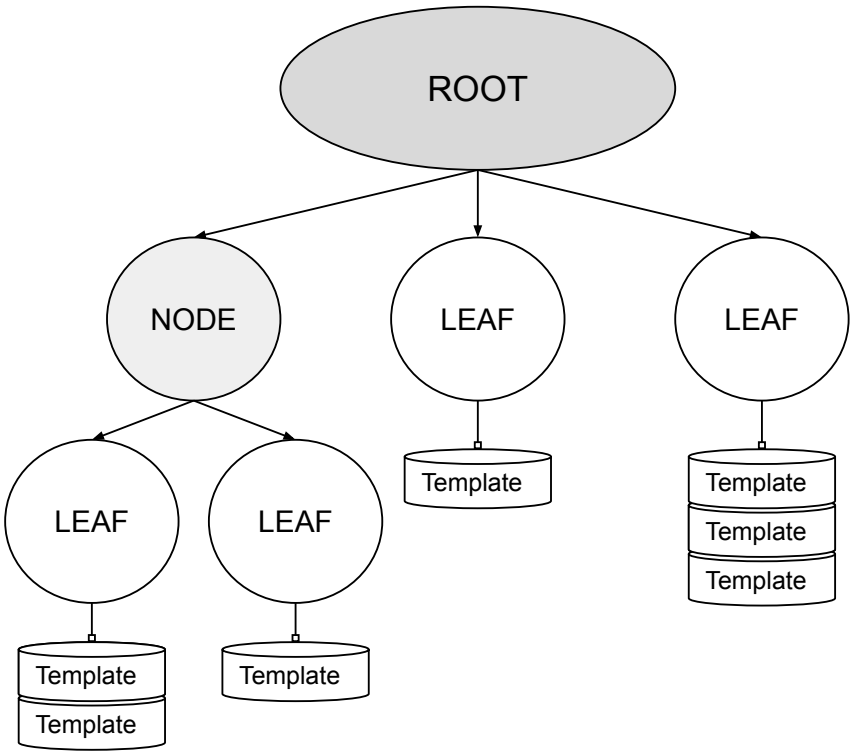


Figure 4.1: USTEP Evolving Research Tree Structure

1. *Root Node* (v_1): At the top of the tree, there is always a root node. It is the only node present during initialization ($\mathcal{V} = \{v_1\}$) and serves as the entry point for search operations.

2. *Internal Nodes* (v_k, where $k > 1$): These nodes have at least one child node and form the backbone of the parsing tree. They help organize and structure the tree.

3. *Leaf Nodes*: Leaf nodes are nodes that do not have any child nodes. They represent the endpoints of the tree branches and serve as the attachment points for templates.

4. *Templates*: Templates are associated only with leaf nodes. Each leaf node can be linked to a maximum of ϕ templates. When a leaf node reaches this template limit, a leaf division operation (explained in Section 4.1.2.4) is triggered to refine the tree structure by utilizing accumulated knowledge.

4.1.2.2 Tree Descent Rules

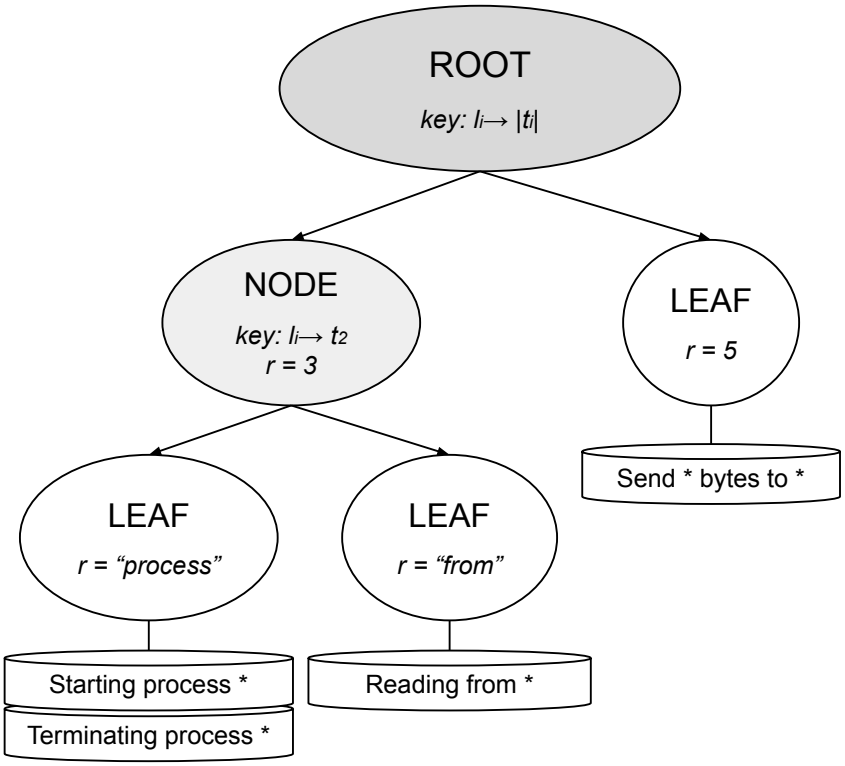


Figure 4.2: Example of USTEP Search Tree

In the USTEP research tree, each leaf node stores a set of discovered templates while root and internal nodes encode parsing rules. The parsing rules in USTEP differ based on the type of node. Root and internal nodes encode parsing rules, while leaf nodes store a set of discovered templates. These rules are utilized during the traversal of the tree to locate or identify a template that matches the given log message.

Each node v_j, which has K_j children, is associated with a key function $v_j.key(\mathbf{t_i})$ and a list of results $r_1, \ldots, r_{K_j}$, where each result r_k corresponds to the k-th child of the node. In the case of the root node v_1, the key function is defined as $v_1.key(\mathbf{t_i}) = |\mathbf{t_i}|$, which represents the number of tokens in the log message. On the other hand, for inner nodes, the key function is $v_j.key(\mathbf{t_i}) = t_{pivot_j}$, where pivot indicates the specific position used for splitting.

Figure 4.2 displays a tree structure with a root node, one internal node, 3 leaves, and 4

templates. The root node parsing rule directs the messages to the left or right part of the tree depending on the value of $|t_i|$. Log messages with $|t_i| = 3$ are directed to the internal node on the left. The internal node parsing rule returns t_2, routing logs towards one of its two leaves, identified by $r_1 =$ "$process$" and $r_2 =$ "$from$".

4.1.2.3 Parsing Algorithm

Algorithm 1 USTEP Parsing Algorithm

```
1: function Parse(t_i)
2:     (leaf_i, (e_i, v_i)) ← SelectClosestLeaf&Parse(t_i)        ▷ Step 1
3:     UpdateTree(leaf_i)                                        ▷ Step 2
4:     return (e_i, v_i)
5: end function
```

Parsing a tokenized log message $\mathbf{t_i}$ using USTEP is a two-step process (Algorithm 1): 1/ Identify the template $\mathbf{e_i}$ and the variables v_i matching $\mathbf{t_i}$ using descent rules; 2/ Refine the tree structure by dividing saturated leaves.

Methods 2 USTEP Template Mining Methods

```
   function InitializeTemplate(t_i)
2:     e ← t_i
       return e
4: end function

6: function UpdateTemplate(e, t_i)
       e ← (e_j if e_j = t_j^i else * : j = 1 ... |t_i|)
8:     return e
   end function
10:
   function getVariables(e, t_i)
12:    v ← (t_j^i : j = 1 ... |t_i| if e_j = *)
       return v
14: end function
```

In Methods 2, we describe three utility functions used to work with log templates. InitializeTemplate to create a template based on an incoming log, UpdateTemplate to refine a template by filtering out position-wise non-identical tokens between the incoming log and an

existing template, and getVariables to extract the variables list from a log once the matching template is known.

$$simF(\mathbf{t_i}, \mathbf{e}) = \frac{\sum_{j=1}^{|\mathbf{t_i}|} equ(t_j^i, e_j)}{|\mathbf{t_i}|} \tag{4.1}$$

$$equ(t, \widetilde{t}) = \begin{cases} 1 & \text{if } t = \widetilde{t} \\ 0 & \text{otherwise} \end{cases} \tag{4.2}$$

Methods 3 Searching methods

```
    function SelectClosestLeaf&Parse(t_i)
2:      leaf_i ← TreeDescent(v_1, t_i)
        (e_i, v_i) ← SelectTemplate(leaf_i, t_i)
4:      return (leaf_i, (e_i, v_i))
    end function
6:
    function TreeDescent(v_j, t_i)
8:      while not v_j.isLeaf() do
            if ∃k : r_k = v_j.key(t_i) for k = 1, ..., K_j then
10:             v_j ← v_k
            else
12:             e_i ← InitializeTemplate(t_i)          ▷ New leaf to include t_i
                Create a new leaf v_j with a templates list [e_i].
14:         end if
        end while
16:     return v_j
    end function
18:
    function SelectTemplate(leaf_i, t_i)
20:     e_i ← argmax{simF(t_i, e) : e ∈ leaf_i.templates}
        if simF(t_i, e_i) > σ then
22:         e_i ← InitializeTemplate(t_i)              ▷ New template to include t_i
            Add new template e_i in leaf_i tempaltes list
24:     else
            e_i ← UpdateTemplate(e_i, t_i)
26:     end if
        v_i ← getVariables(e_i, t_i)
28:     return (e_i, v_i)
    end function
```

The main search function is SelectClosestLeaf&Parse (Methods 3). As templates are attached to leaves, to assign a template to a log t_i, the first step is descending the tree using embedded rules. Once a leaf is reached, the list of associated templates is scanned to pick the one matching t_i, if none apply, a new template is created based on t_i and added to the tree structure (Lines 12 and 22). The similarity scores between a log and the templates attached to the selected leaf are computed as described in 4.1. $simF(\mathbf{t_i}, \mathbf{e})$ represents the proportion of positions-wise identical tokens between $\mathbf{t_i}$ and a template e. The template with the highest similarity score above σ is picked to represent the log. If no score above σ is found a new template is created using the current log.

4.1.2.4 Tree Update Operations

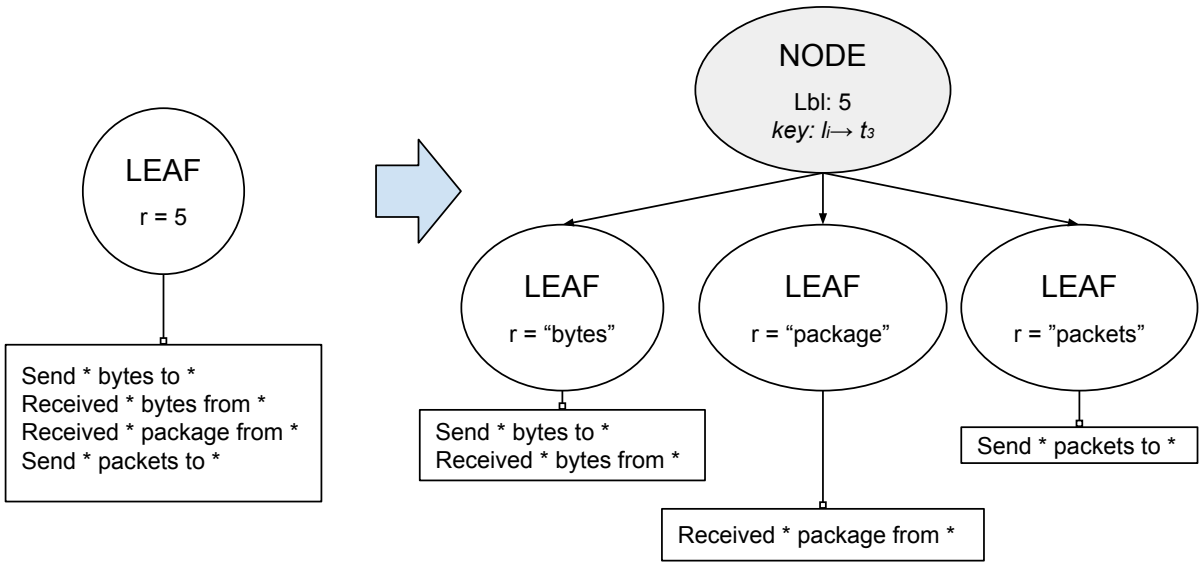


Figure 4.3: Example of Leaf Division with Pivot set to 3

Parsing time can be a bottleneck for downstream applications. To optimize the processing time of the next parsing operations, tree update operations are triggered after parsing a log if the returned leaf is linked to more than ϕ templates. The saturated leaf is transformed into an internal node and the templates are dispatched among new leaves. This step limits the number of similarity factors to compute for parsing a single log line and directly contributes to USTEP

constant parsing time (Section 4.2.5.1).

Methods 4 Tree Update Methods

```
    function UpdateTree(leaf_i)
2:      if |leaf_i.templates| > φ then
            DivideLeaf(P, leaf_i)
4:      end if
    end function
6:
    function DivideLeaf(leaf)
8:      pivot ← SelectPivot(leaf)
        Create an inner node v with
10:             a pivot position pivot and an empty list of leaves L
        for all e_i ∈ leaf.templates do                ▷ Share templates between new leaves
12:         if ∃x ∈ L : e^(j)_pivot = e^(i)_pivot ∀e_j ∈ x.templates then
                Append e_i to template list of x
14:         else
                Create a new leaf with template e_i and append to L
16:         end if
        end for
18:     Replace leaf by inner node v in P
    end function
20:
    function SelectPivot(leaf)
22:     Let's length templates length of leaf
        for p ← 1, ... length do
24:         S_p ← {e^(i)_p : e_i ∈ leaf.templates}          ▷ Set of tokens at position p
        end for
26:     pivot = argmax{|S_p| : p ∈ 1, ... length }
        return pivot
28: end function
```

UpdateTree algorithm and the related methods are described in Methods 4. The leaf division process starts by searching for the position with the highest number of different tokens among the *templates*. This position p is set to be the pivot. The leaf is then transformed into an internal node with the same label and a key function $key(l_i) = t_p$. For each unique token at position p within the templates, a new leaf is created with the appropriate label. All the templates attached to the initial leaf node are transferred to a newly created leaf following the new descent rule.

Figure 4.3 illustrates the division process for $\phi = 3$. The considered leaf has 4 templates and is therefore saturated. The position with the highest number of unique tokens is $p = 3$ with 3 different ones (*bytes*, *packages*, *packets*). Using this pivot, the leaf is transformed into an internal node with $key = t_3^i$, and three new leaves are created, one for each unique word at p. The existing templates are dispatched to the new leaves according to their token value at position 3.

4.2 A Comparative Evaluation of Online Log Parsers

To evaluate the relevance of USTEP regarding the existing online parsing algorithms, we conducted a theoretical and an experimental evaluation of its accuracy, robustness and efficiency [31].

4.2.1 Experimental Setup

4.2.1.1 Datasets

Name	#log	size(B)	#template	#t/line	%var
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.
Big-comp	2×10^9	150 GB	910	6.6	33%

Table 4.1: Datasets Characteristics

We selected 14 datasets coming from a wide panel of real-world applications: 12 comes from openly accessible online sources [110], 1 from *3DS OUTSCALE* infrastructure (*Internal-*

1), and 1 was crafted by combining multiple log traces (*Big-comp*). *3DS OUTSCALE* dataset represents 30 minutes of consecutive logs issued only from sources related to TINA OS the 3DS OUTSCALE Cloud orchestrator. *Big-comp* was crafted using Loghub [110] open source datasets and represents a multi-source logging environment which is to be expected for a cloud computing platform. The 12 other datasets are extracted from the Loghub platform [110], a bank of openly accessible log traces. We summarize the datasets characteristics in Table 4.1. In some cases, the ground truth is unknown (*HDSF-extended*, *OpenStack-extended*, *3DS OUTSCALE*), so, some metrics cannot be calculated.

4.2.1.2 Log Parsers

Log parser	Year	Method	#parameters
Drain	2017	Fixed depth parsing tree	3
LenMa	2016	Clustering	1
SHISO	2013	Clustering	4
Spell	2016	Longest common subsequence	1
USTEP	2021	Evolving parsing tree	2

Table 4.2: Log Parser Characteristics

To prove the effectiveness of USTEP, in this section, we compare its performance with four state-of-the-art online log parsers in terms of accuracy and efficiency.

- Drain [103] is an online log parser based on a fixed-depth research tree structure. It is used by many log-based applications [24; 29] and is presented as the best parsing solution within Zu & al benchmark [1].
- LenMa [101] uses the distribution patterns within token length to group logs in an online fashion.
- SHISO [100] uses a tree with a predefined number of children per node to guide its log grouping process.

- Spell [102] is an online log parser based on the Longest Common Subsequence algorithm (LCS). It uses different pre-filtering techniques to accelerate the matching process.

The principal characteristics of each method are presented in Table 4.2. Each log parser uses different parameters (e.g., σ, ϕ for USTEP), for fairness of comparison each presented experiment was run multiple times to fine-tune the parameters and retain only the best value.

4.2.1.3 Additional Tuning

Dataset	σ	ϕ	Dataset	σ	ϕ
Apache	0.5	4	Internal-1	0.3	50
BGL	0.45	18	Mac	0.6	18
Big-comp	0.3	8	OpenSSH	0.75	3
Hadoop	0.3	3	OpenStack	0.9	2
HDFS	0.35	2	OpenStack-2	0.9	2
HDFS-2	0.35	2	Thunderbird	0.3	4
HPC	0.6	2	Z.keeper	0.63	3

Table 4.3: USTEP Parameters Values

The experiments we present here were carried out on a CentOS Linux 7.8 Cloud Hosted virtual machine with 32 hearts, and 62 GB of RAM. We have made the implementation of each log parser used publicly available and linked to the datasets. However, due to security and confidentiality concerns, we have kept Internal-1 private.

The considered algorithms process logs online, and for fairness of comparison, we applied the same processing rules to each log parser. Used values of σ and ϕ for each dataset are summed up in Table 4.3. For space reasons, we didn't detail here the settings for all the other methods but they are available with the source code of USTEP[1].

4.2.2 Effectiveness of the Log Parsers

Traditionally log parsing is evaluated using the Parsing Accuracy (PA)[1; 102], which corresponds to the ratio of correctly parsed log messages over the total number of log messages. A

[1]https://github.com/outscale/ustep-online-log-parser

Dataset	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**
Average	0.903	0.855	0.707	0.810	**0.937**
Big-comp	0.480	-	-	-	**0.816**

Table 4.4: Parsing Accuracy of Log Parsers

log message is considered correctly parsed when it aligns with the same template as its ground truth. Given the potential impact of errors on the performance of downstream applications, accuracy is a crucial aspect. For this study, we employed our ten labeled datasets. The parameters of all the log parsers are fine-tuned through over 100 runs to avoid bias from randomization, and we reported the best result for each dataset in Table 4.4.

Our proposal method, USTEP exhibits great performance, achieving the highest PA on eight out of ten datasets and closely approaching the best result on the remaining two. Overall, USTEP has the best average PA at 0.937, 3.4% higher than the second best (Drain with 0.865). USTEP's better results are due to several intrinsic characteristics. First, the leaf division process allows it to choose the best token as a descend rule. Where Drain always selects the n-th token with the risk of selecting a yet-undiscovered variable leading to a branch explosion, splitting logs from the same log statement over different leaves. Besides, the τ parameters allow fine-tuning over different types of datasets.

USTEP and Drain appear to be the on average two most accurate methods on the considered datasets. Production environments are often more complex and englobe a large range of logging sources. *Big-comp* dataset was crafted using parts of each of the other 10 datasets to represent

such an environment. USTEP achieves a 0.816 parsing accuracy over *Big-comp* dataset, this is 0.336 more than Drain (0.480). We believe this significant difference to be due to the way both algorithms manage their tree structures. Drain uses the *depth*-first tokens as descent rules, if a non-numeric variable is part of those, it will create a new leaf for each unique value it takes. On the other side, USTEP descent rules are crafted thanks to the accumulation of templates allowing it to limit the case where a variable sensitive position is selected for a descent rule allowing it to handle more diverse datasets.

4.2.3 Robustness of the Log Parsers

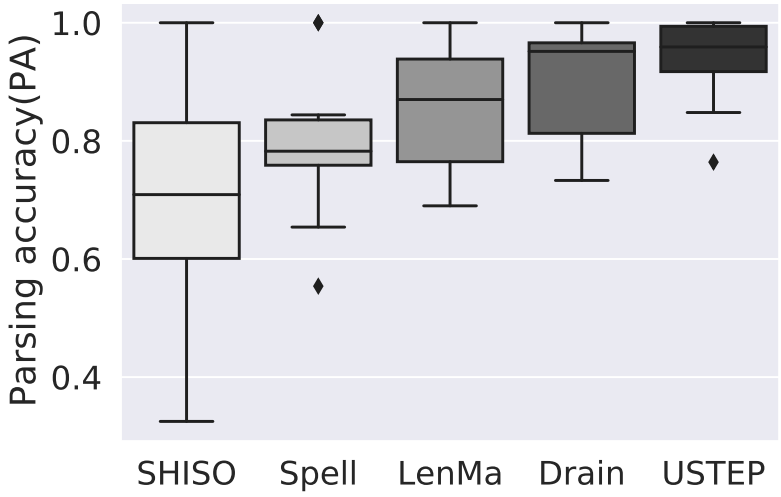


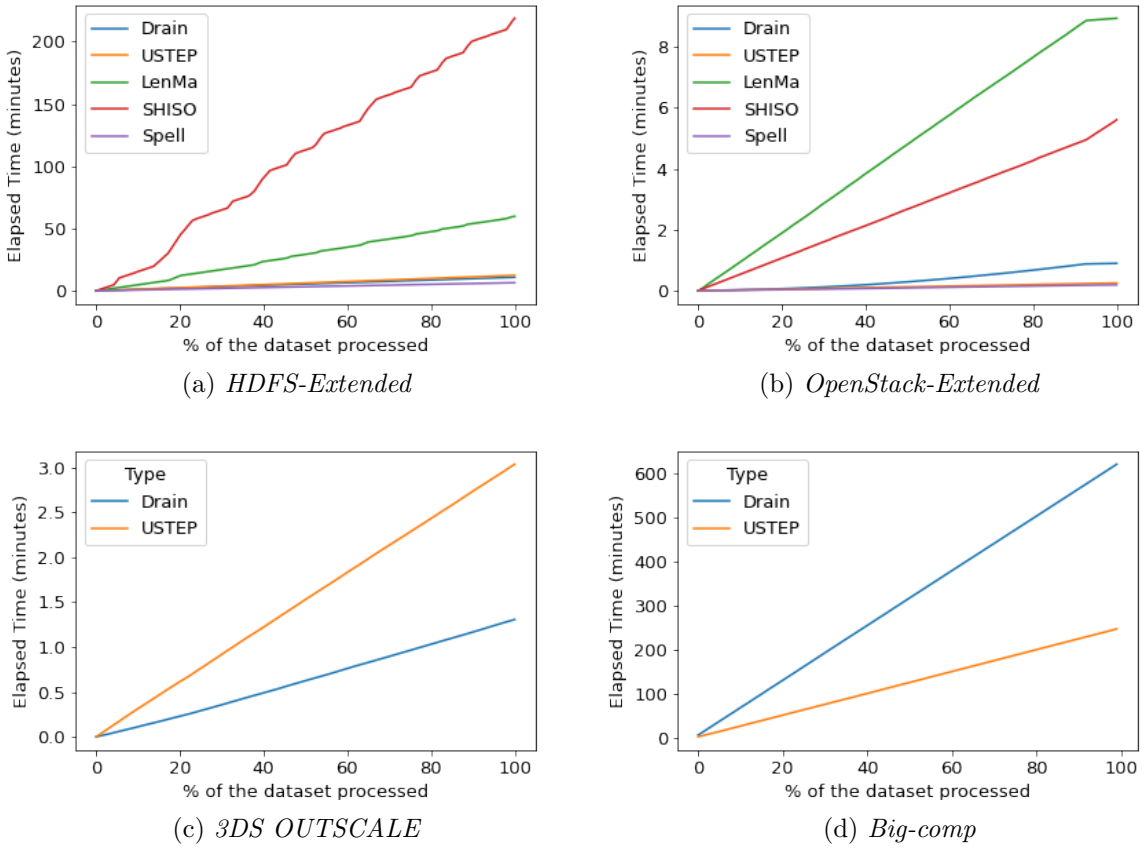
Figure 4.4: Robustness of Log Parsers

We believe that a general parsing method must deliver robust performance on diverse datasets for it to be utilized in production environments. Our goal here is to illustrate USTEP's capacity to support a broad range of log data types. In Figure 4.4, we plot the accuracy distribution of each method. The log parsers are arranged in ascending order of the median PA. SHISO is on the lowest side and USTEP is on the highest one. Achieving consistently high PA values across a diverse range of datasets is critical for the general use of a log parser. Regarding this, despite their great average PA (Table 4.4), the considered log parsers have a large variance when

applied to logs coming from different types of logging environments. The 1st to 3rd quartile distance for SHISO is 0.23, 0.146 for Drain, 0.079 for Spell, and 0.073 for USTEP which also has the highest median at 0.959 PA. The robustness of USTEP within diverse environments is supported by its achieved PA on Big-comp dataset.

4.2.4 Efficiency of the Log Parsers

4.2.4.1 Experimental Parsing Time



(a) *HDFS-Extended* (b) *OpenStack-Extended*

(c) *3DS OUTSCALE* (d) *Big-comp*

Figure 4.5: Cumulated processing time by dataset

We plotted the cumulative time required by each method to parse the four largest datasets (*HDFS-Extended*, *Openstack-Extended*, *3DS OUTSCALE*) and *Big-comp* (Figure 4.5). Drain and USTEP were the only methods able to process the *3DS OUTSCALE*, and *Big-comp* datasets

in less than a day. The effectiveness of each method varies with the dataset; however, both present a linear cumulated parsing time on all the datasets.

4.2.5 Memory Usage of Log Parsers

Table 4.5: Memory Usage in MB

Dataset	Drain	LenMa	SHISO	Spell	USTEP
Apache	1.917	1.796	1.714	2.007	1.784
BGL	3.777	3.987	3.581	3.878	3.687
Hadoop	3.079	3.095	2.875	3.228	3.035
HDFS	2.835	2.738	2.636	2.957	2.712
HPC	2.700	2.704	2.479	2.753	2.598
Mac	3.368	3.732	3.161	3.456	3.442
OpenSSH	2.734	2.637	2.541	2.854	2.622
OpenStack	4.004	4.009	3.769	4.342	4.233
Thunderbird	4.053	4.062	3.812	4.151	4.058
Zookeeper	2.988	2.926	2.795	3.035	2.894

Considered parsing methods are based on different parsing structures and therefore can have a different memory footprint. In Table 4.5, we have consigned the memory used in MB by each parsing method after parsing a dataset. The results obtained show no significant deviation in the memory used by the methods. Used space seems to be most impacted by the number of underlying templates as well as the number of tokens in the logs of the dataset rather than by the parser memory structure.

4.2.5.1 Complexity Analysis

We analyze the worst-case time and space complexity to process a single log for Drain, Spell, LenMa, SHISO and USTEP.

Theorem 1. *Let's denote by* $\mathbf{t_i}$ *an incoming tokenized log,* $|\mathbf{t_i}|$ *the number of tokens constituting* $\mathbf{t_i}$*,* $|\mathbf{t}|$ *the highest number of tokens for a single discovered template,* T *the number of unique templates already discovered by the parser using a structure with* N *nodes. We have the time and space complexities in the worst-case scenarios summarized in Table 4.6.*

Table 4.6: Worst-case complexities

Log parser	Searching time complexity	Space complexity
USTEP	$\mathcal{O}(\lvert\mathbf{t_i}\rvert)$	$\mathcal{O}(T \times N)$
Drain	$\mathcal{O}(T \times \lvert\mathbf{t_i}\rvert)$	$\mathcal{O}(T \times N)$
Spell	$\mathcal{O}(T \times \lvert\mathbf{t_i}\rvert \times \lvert\mathbf{t}\rvert)$	$\mathcal{O}(T)$
LenMa	$\mathcal{O}(T \times \lvert\mathbf{t_i}\rvert)$	$\mathcal{O}(T)$
SHISO	$\mathcal{O}(T \times \lvert\mathbf{t_i}\rvert)$	$\mathcal{O}(T \times N)$

Proof. We summarize the parser structures to determine complexities. USTEP and Drain use searching tree structures to determine the closest leaf, once the leaf is identified, a list of templates is evaluated to find the most similar. So, both have the same space complexity $\mathcal{O}(T \times N)$ and a searching time complexity $\mathcal{O}(d + \max\{c_{\text{leaf}}\ \forall\text{leaf}\} \times \lvert\mathbf{t_i}\rvert)$ where d is the tree depth and $\max\{c_{\text{leaf}}\ \forall\text{leaf}\}$ is the maximum number of candidate templates in leaves. In the worst-case scenario, Drain attaches all discovered templates to the same leaf having $\max\{c_{\text{leaf}}\ \forall\text{leaf}\} = T$, however, USTEP bounds this number by the parameter ϕ. In both cases, the depth of the tree doesn't exceed the maximum token length $\lvert\mathbf{t}\rvert$. USTEP searching time complexity is therefore in $\mathcal{O}(\lvert\mathbf{t_i}\rvert + \phi \times \lvert\mathbf{t_i}\rvert)$, with ϕ a constant. Spell does not partition its search space and will use the LCS algorithm to compare t_i with all the existing templates. LCS algorithm complexity is in $\mathcal{O}(\lvert t_i\rvert \times \lvert\mathbf{t}\rvert)$. Thus Spell time complexity is in $\mathcal{O}(\lvert t_i\rvert \times \lvert\mathbf{t}\rvert \times T)$. LenMa starts by iterating over the log message to create a word length vector representing the character length of each token. It then compares this vector, with all the discovered vectors with the same length ($\lvert t_i\rvert$). Therefore, its time complexity is in $\mathcal{O}(\lvert t_i\rvert + T \times \lvert t_i\rvert)$. In the worst case, SHISO tree descent algorithms can visit all the existing nodes in the structure and compare the log with all the templates. Its worst-case time complexity is then in $\mathcal{O}(T \times \lvert t_i\rvert)$.

□

Search tree structures are more memory-intensive but help to reduce processing time. For instance, while Spell exhibits a linear space complexity with respect to the discovered templates, its time complexity is quadratic in terms of the maximum token length. On the other hand,

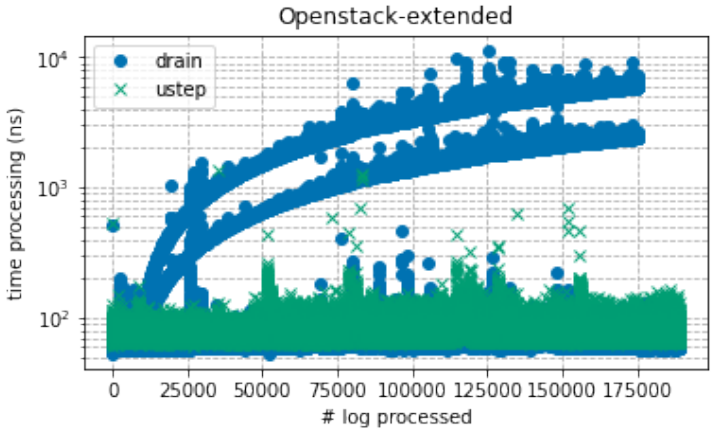


Figure 4.6: Processing Time Evolution for OpenStack-extended

USTEP presents a larger space complexity but it is the only method with a constant worst-case time complexity.

For *OpenStack-extended* dataset, Drain performs worse than USTEP. Figure 4.6 which illustrates the processing time for each incoming log, depicts a trend of increasing processing times for Drain. As Drain doesn't bound the number of templates attached to a leaf, in the long run this increases the processing time as more templates are discovered. USTEP maintains a more stable processing time thanks to ϕ parameter that limits the number of similarity comparisons.

4.3 USTEP-UP

The time complexity analysis of USTEP displays demonstrates a consistent parsing time relative to the number of tokens inside the message. Results of our experimental evaluation support this claim. However, within fast-paced ou multi-sources logging environments, even a constant parsing time can be a limitation for real-time parsing. We address this scalability issue, by presenting in this section USTEP-UP, a way of running multiple slightly modified USTEP-like instances in parallel.

Agrawal & al [107] used an LCS-based log parsing algorithm similar to Spell and distributed it on top of Spark. Still, their evaluation displays a lower bound in parsing time due to the

increased overhead induced by the knowledge merge steps between the instances. A similar approach applied to Drain would run into an even bigger problem, as it's more than likely that the tree structures of the instances will diverge, leading to inconsistent parsing where different templates can be associated with the same log message depending on the instance parsing it. This inconsistency is a problem because identical messages could appear as being different for downstream applications.

Like Drain instances, USTEP ones are stateful and need to share information; otherwise, they might encode different parsing rules. To address this, USTEP-UP runs modified USTEP instances. Those skip the leaf division step and never divide saturated leaves. A monitoring instance is added to the architecture, it spreads the knowledge and refines the instances by punctually merging its memory structure with theirs.

4.3.1 Architecture

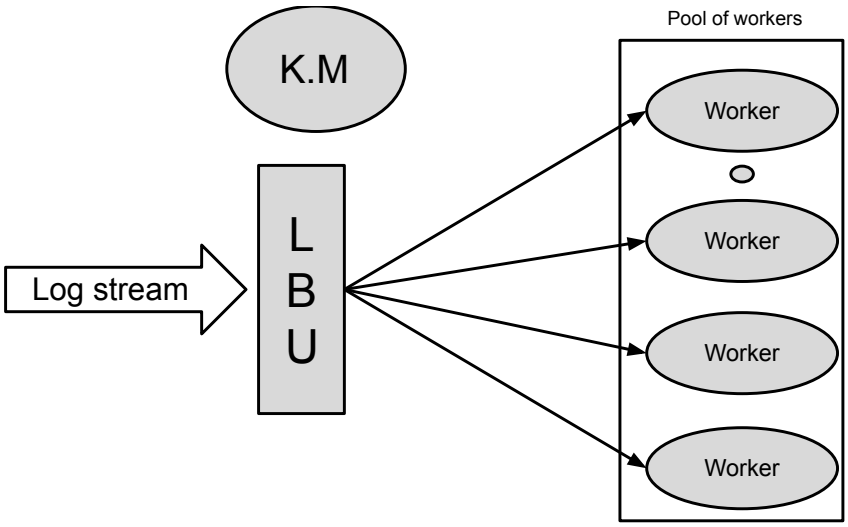


Figure 4.7: USTEP-UP architecture

USTEP-UP architecture (Figure 4.7) is designed to run multiple USTEP-like instances in a decentralized fashion. We will refer to those N parsing instances as $\mathcal{U}^l = \{\mathcal{P}^l, \sigma, \phi\}_{l=1}^{N}$. There is no interaction between any of those parsing instances, they receive their workload from a Load Balancer Unit (LBU) and punctually merge their knowledge with the Knowledge Manager(KM).

Subsequently, we will refer to it as $\mathcal{U}^0 = \{\mathcal{P}^0, \sigma, \phi\}$. $\mathcal{U}^0$, never processes logs, it updates its tree $\mathcal{P}^0$ by merging it with other trees ($\mathcal{P}^l$), the resulting tree also replaces the $\mathcal{P}^l$. Initially, every worker starts with identical parsing trees, before any parsing operations: $\mathcal{P}^0 = \mathcal{P}^l, \forall l$.

As they process logs, those trees will grow and extend their parsing knowledge. To avoid ending with conflictual parsing rules (descent rules) within two trees, we run a modified USTEP version that skips the leaf division step and therefore never splits saturated leaves. Bypassing this step ensures that no *internal node* is created by a parsing instance, it also influences the time complexity of modified instances now in $\mathcal{O}(maxT \times |\mathbf{t_i}|)$(linear) against $\mathcal{O}(|\mathbf{t_i}|)$(constant) for classic USTEP instances. With $maxT$ the maximum number of templates attached to one leaf. The main risk is then a continuous slowdown of instances with the increase in the number of log templates. To avoid this, it is the knowledge manager that divides saturated leaves when it merges its tree with an instance one. Delegating the split step to one instance guarantees that the encoded parsing rules won't diverge.

Load balancing systems are widely studied, and many different approaches have already been detailed within the literature[115; 116]. As its intrinsic workflow doesn't affect USTEP-UP design, we will not detail any LBU here. If you want to work with this architecture, we recommend using a state-of-the-art load balancer.

4.3.2 Merging knowledge

Delegating the leaf division process only to K.M. assures that the instances trees won't discover diverging parsing rules. As the worker's instances can't divide leaves, it is guaranteed that at any moment, any internal nodes in a tree $\mathcal{P}^l$ also exist in $\mathcal{P}^0$. To distribute the knowledge and refine the instance tree structures, K.M. punctually merges its tree $\mathcal{P}^0$ with a selected instance tree $\mathcal{P}^l$, the resulting tree replaces both $\mathcal{P}^0$ and $\mathcal{P}^l$. This merging process adds the knowledge collected by the instance to K.M., but it also distributes the knowledge acquired by K.M. toward the other instances to $\mathcal{U}^l$.

The merging process (Algorithm 5) starts by pointing at the root node of each tree. It uses

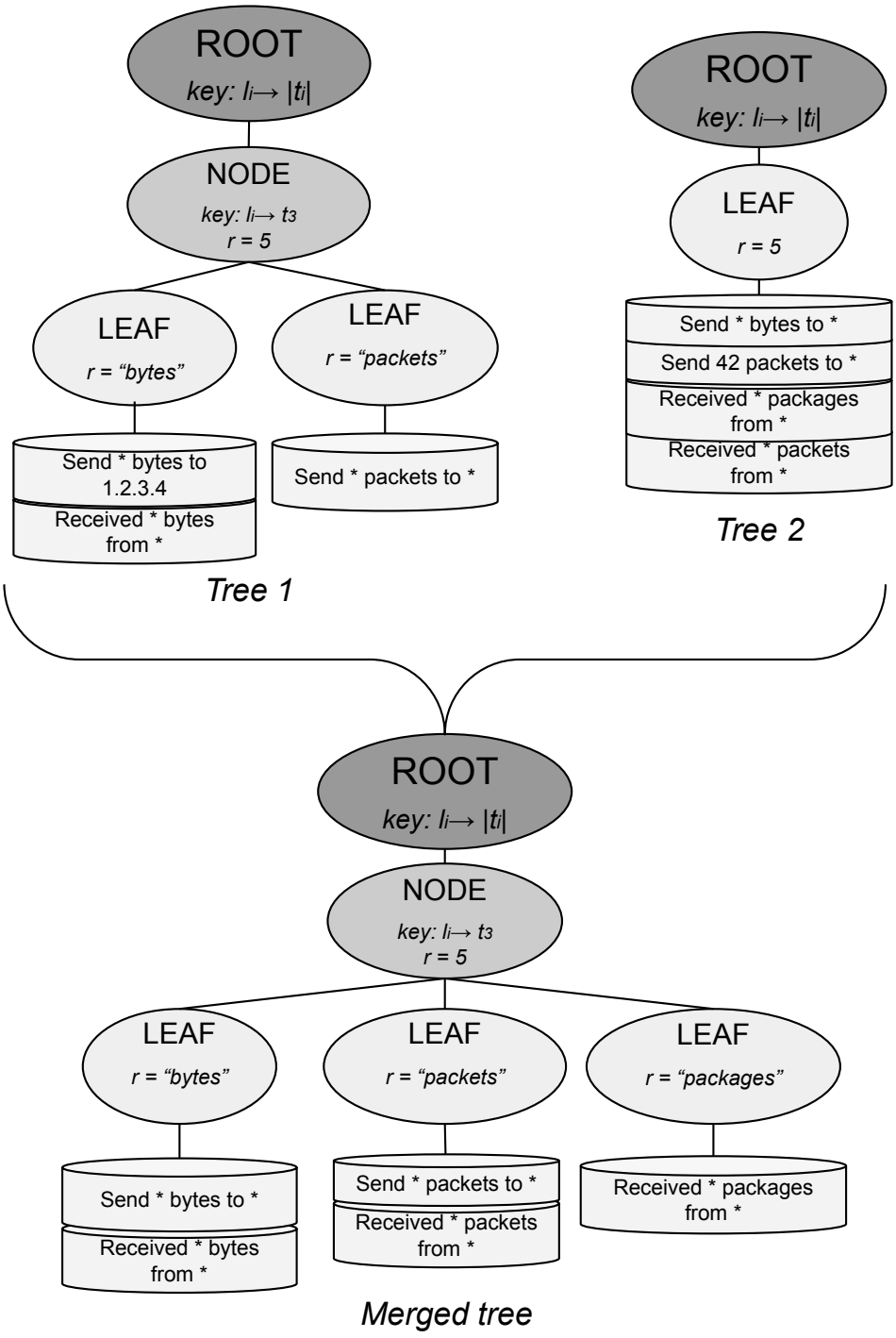


Figure 4.8: Example of merging tree process

Algorithm 5 Merge $\mathcal{P}^l$ With $\mathcal{P}^0$

```
1: function MergeTree(P^0, P^l)
2:     q ← FIFOQueue()
3:     q.enqueue(v_1^0, v_1^l)
4:     while not q.isempty() do
5:         (node1, node2) ← q.dequeue()
6:         for child in node2.childrens do
7:             if not child.isLeaf() then
8:                 q.enqueue((node1.getChild(child.label), child))
9:             else
10:                if child.label in node1.childrens() then
11:                    for template in child.templates() do
12:                        node1.parse(template)
13:                    end for
14:                else
15:                    node1.childrens.add(child)
16:                    if node1.getChild(child).isSaturated() then
17:                        node1.getChild(child).divide()
18:                    end if
19:                end if
20:            end if
21:        end for
22:    end while
23:    return P^0
24: end function
```

a First in First Out (FIFO) Queue to iterate over all the nodes of $\mathcal{P}^l$ in a breadth-first fashion. Crossed *Internal nodes* are enqueued paired with their equivalent in $\mathcal{P}^0$. When encountering a *leaf node*, if no child with the same label exists in $\mathcal{P}^0$, the leaf is added to the currently pointed node in $\mathcal{P}^0$. Otherwise, all the *templates* from the $\mathcal{P}^l$ leaf are parsed through $\mathcal{P}^0$ starting at the current node and not at the root node. MergeTree($\mathcal{P}^0, \mathcal{P}^l$) function time complexity is in $\mathcal{O}(|\widetilde{\mathsf{E}}|)$, as we need to iterate over every template within $\mathcal{P}^l$ and parse them with $\mathcal{P}^0$. As this function performs operations in place, its space complexity is in $\mathcal{O}(|\mathcal{V}^0| \times |\widetilde{\mathsf{E}}^0| + |\mathcal{V}^l| \times |\widetilde{\mathsf{E}}|)$.

Figure 4.8 illustrates the merging process between the knowledge tree $\mathcal{P}^0$ (Tree 1) and an instance tree $\mathcal{P}^l$ (Tree 2). In this example, we consider $\sigma = 0.5$ $\phi = 3$. The process starts by visiting (v_1^0, v_1^l), the two root nodes. In this example, v_1^l is linked to one leaf node v_1^1, and $v_1^1.label = 5$. As there is already a node with $node.label = 5$ inside $v_1^0.children$, all the templates

attached to v_1^1 are parsed by v_1^0. $v_1^1.templates$ return four elements: 1/ *"Send * bytes to *"* will end up in the left leaf according to the descent rules as simF(*"Send * bytes to *"*, *"Send * bytes to 1.2.3.4"*) = 0.8, it will update the template within $\mathcal{P}^0$.2; 2/ *"Send 42 packets to *"* will be affected to the already more refined template *"Send * packets to *"*; 3/ *"Received * packages from *"* will create a new leaf as no leaf with *label* = "*package*" exists yet. 4/ *"Received * packets from *"* will end up in the same leaf as *"Send * packets to *"* as a new template.

4.4 Perspectives

By separating variables from the template, the parsing process opens a new source of information for intelligent monitoring, and it is a required step for log-based anomaly detection methods. With the proposition of USTEP - a high-performing online log parsing algorithm in terms of accuracy, robustness, and effectiveness, along with USTEP-UP, an architecture designed to run multiple parallel USTEP instances, we effectively tackle the challenge of parsing logs in environments characterized by high log statement evolution and huge data volume.

One limitation of USTEP is that it can't group logs sharing the same template but with a different number of tokens. This limitation is inherent to its memory structure and it's shared with Drain. Still, our comparative evaluation on 14 datasets displays that USTEP outperforms existing methods in terms of PA and robustness. We believe that progress in the parsing field can stimulate progress in many log-based applications and not just the anomaly detection ones as it allows using log message embedded information for a wide variety of tasks. Also, we discovered that parsing algorithms like USTEP can end up having unusual applications such as the classification of JIRA[1] tickets or the extraction of invoice-embedded information by some 3DS OUTSCALE teams.

[1] https://www.atlassian.com/fr/software/jira

CHAPTER 5

Monilog: An Automated Log-Based Anomaly Detection System

Contents

5.1 Introduction

In this chapter we present Monilog, a log-based alert system architecture designed to detect anomalies at a cloud scale. It is able to produce comprehensive reports regarding the reported anomaly including the concerned system and applications. Unlike existing methods depicted in Chapter 2, Monilog operates without needing specific knowledge of the relationship between logs, thereby facilitating near real-time alerts through a streaming mechanism.

We start by exploring the architecture of Monilog and our work toward implementing Monilog inside 3DS OUTSCALE infrastructure. We successfully used this implementation to monitor equipments from the European region of 3DS OUTSCALE during 11 consecutive days. Details about this experimentation, its context, and the obtained results are to be found under Chapter 6.

5.2 Monilog Architecture

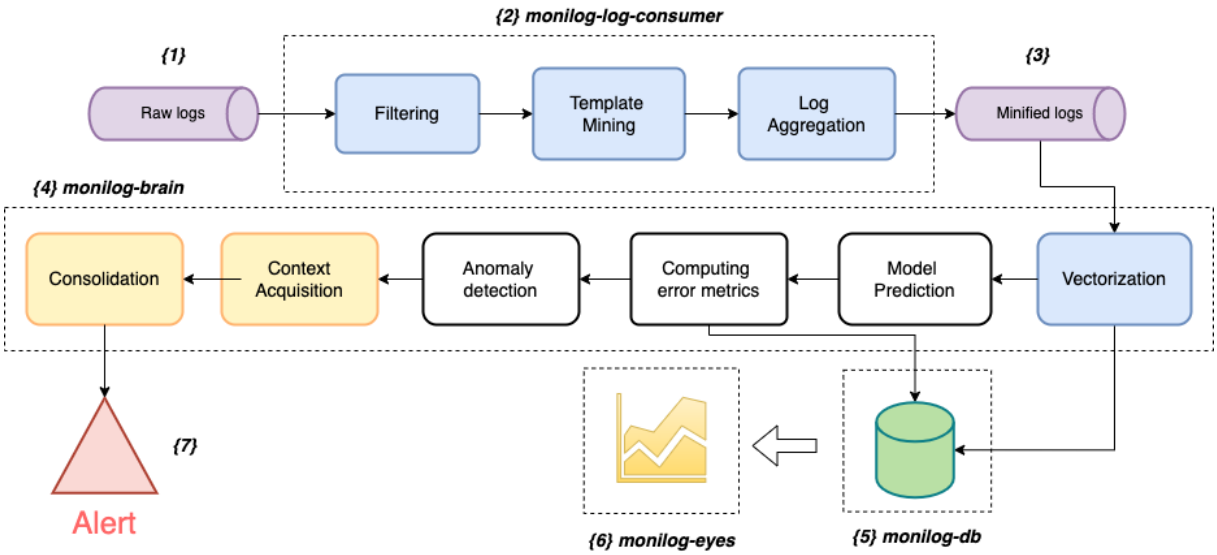


Figure 5.1: Monilog Architecture and Workflow

Monilog workflow is divided in three parts (Figure 5.1): (Section 5.2.1) First (in blue), Monilog infers log templates and vectorizes them; (Section 5.2.2) It seeks anomalous events and

generates alert candidates (in white); The last step (in yellow), consolidates alert candidates by grouping related ones and acquiring their context (Section 5.2.3). On the architectural schema (Figure 5.1), we indicate 7 components enumerated from 1 to 7, within this chapter, we will refer to those using a number within curly braces (e.g. {1} refers to the raw logs message queue).

5.2.1 Vectorizing Logs

5.2.1.1 Log Filtering

Logs are timestamped and generated by machines at runtime. Monilog input({1}) is an infinite and temporally ordered sequence of tuples $\mathcal{L} = ((t_i, l_i), i = 1, 2, ...)$, with t_i the timestamp associated to a log l_i. In our implementation of Monilog, this input {1} is a Kafka[1] stream exposing the last 3 hours of logs. With Kafka being a distributed event streaming platform presenting good performance in terms of scalability and fault tolerance.

In first-hand, logs are filtered based on header embedded criteria ({2}). Filtering is an optional step that allows you to retain only the logs linked to the system you want to monitor. For instance, in our experimental use case around serves crashes (Chapter 6), we choose to retain only logs issued by Kernel Virtual Machines from 3DS OUTSCALE European cloud region.

5.2.1.2 Template Mining

We define the log event template (e) to be the combination of the application name and the severity of the considered log message. For instance, the template of the raw log presented in Figure 1.3 would be *"serviceManager_INFO"*. This embedding bounds the number of different possible templates by the number of running applications multiplied by the number of distinct severity levels (8 for syslog formatted logs). Also, it keeps track of the related application and the severity, this property is exploited by the context acquisition step to generate contextual anomaly reports.

In Chapter 4, we presented our proposal log parsing method USTEP that outperforms ex-

[1] https://kafka.apache.org

isting log parsers in terms of accuracy and robustness. However, USTEP still makes parsing errors and rarely achieves 1.0 PA on log datasets. To avoid dealing with falsely parsed templates we choose to craft templates using only header-embedded information as this is a trivial and non-error-prone operation. Log parsers can generate useful information based on the log messages. We believe that in the future, replacing the proposed template mining step by a log parser such as USTEP could improve the system performances, although this requires dealing with the induced potential parsing errors.

5.2.1.3 Log Aggregation

Logs with the same host and template are aggregated by w nanoseconds windows. We define $L_{h,e}(t)$ as the count of logs associated to template e for host h at timestamp t, and the logs count window as a time serie $C_{h,e}(w)$.

$$C_{h,e}(t) = \sum_{i \in [t, t+w]} L_{h,e}(i) \tag{5.1}$$

The tuples of size 4 are then queued within another message queue {3} following the shape $(t + w, h, e, C_{h,e}(t))$. Messages queued into {3} are to be consumed by *monilog-brain* {4} component, in charge of identifying alert candidates within the log series associated to each host.

We separate the logic of the *monilog-log-consumer* component {2} and *monilog-brain* component {4}. By doing so, the structure becomes more modular and it is easier to enforce new filtering rules or to experiment with different anomaly detection models. In our practical implementation, the *monilog-log-consumer* is implemented using Spark[1], an engine for executing computing tasks on a cluster of machines. Having its logic decoupled from the *monilog-brain* component allowed us to take full advantage of Spark streaming features.

[1] https://spark.apache.org

5.2.1.4 Vectorization

monilog-brain {4} component reads the message queue {3}, and writes the obtained tuples in a cold storage database {5}. This database aims to provide a convenient way of replaying specific sequences to test new models or to inspect suspicious results. In our implementation of the Monilog system, we choose InfluxDB[1], a high-speed read and write database designed for time series. We also built a visualization module {6} on top of {5}, allowing us live monitoring of interesting metrics such as the log count per time window or the time delay or the prediction error of the different models over time.

When training models, every unique log template within the training set is associated to an id (see template mining step). We define $\mathcal{T}$ as the ensemble of unique log templates within the model training set, and $f_e : \mathcal{T} \mapsto 1, 2, .., |\mathcal{T}|$, a bijective function. The id e_i of log template is defined by the following equation:

$$e_i = \begin{cases} f_e(i) & \text{if } i \in \mathcal{T} \\ |\mathcal{T}| + 1 & \text{otherwise} \end{cases} \tag{5.2}$$

The event count vector $V(t)$ is constructed as follows:

$$V(t)(i) = \sum_{e_i=i} C_{h,e}(t) \tag{5.3}$$

$$|V(t)| = |\mathcal{T}| + 1 \tag{5.4}$$

5.2.2 Detecting Anomalies

5.2.2.1 Model Predictions:

Models forecast the log traffic using a matrix regrouping r previous observations. Their output is a prediction for the next log count vector, each model can be assimilated to a prediction

[1] https://www.influxdata.com

function P:

$$P(M(t)) = \hat{V}(t) \tag{5.5}$$

$$|V(t)| = |\hat{V}(t)| \tag{5.6}$$

With M(t) a matrix composed of the r last count vectors at time t. M(t) is formally defined as follows:

$$M(t)(j) = V(t - (w \times (r - j))), j \in [1, .., r] \tag{5.7}$$

In the case of a perfect predictor:

$$\hat{V}(t) = V(t + w) \tag{5.8}$$

5.2.2.2 Compute Forecasting Error

Previous steps model prediction $\hat{V}(t - w)$ are compared to the actual observation $V(t)$ using different error metrics. They evaluate the forecasting accuracy of the models for the elapsed time step. The performance of Monilog is heavily impacted by the selected model/metric combination. In our experimental evaluation phase, we used three forecasting models, and three error metrics resulting in 9 different error time series, one for each model/metric pair. We found useful to store error metric outputs into the cold-storage database {5}. This allows us to visualize the error series in real time but also to investigate again certain patterns later without having to repeat the model prediction step, which is expensive in terms of computing units.

5.2.2.3 Alert Candidate Generation

The error time series are used to generate alert candidates. Points that are considered abnormal regarding the three sigma rules based on the weekly mean error are labeled as alert candidates.

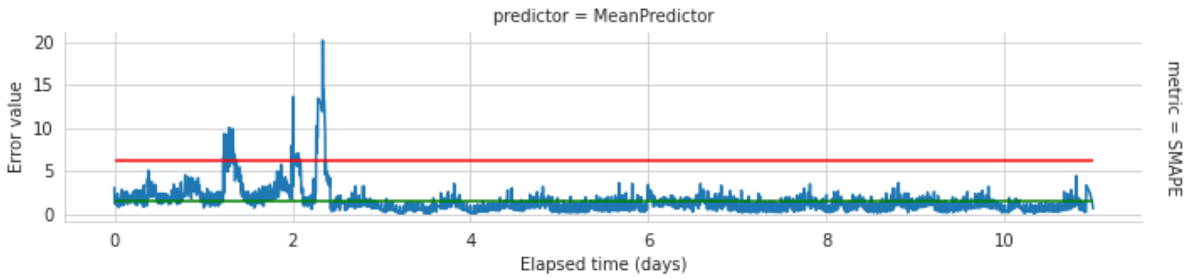


Figure 5.2: An Example of Error Time Serie

Figure 5.2 is an error-time series extracted from our experimental evaluation (Section 6.2). It plots the error value (vertical axis) over the time (horizontal axis), the green line represents the mean value for the serie, and the red one the 3-sigma threshold. Any point above this threshold will be considered as an alert candidate.

5.2.3 Anomaly Consolidation

5.2.3.1 Context Acquisition

Forecasting error metrics are computed using the forecasted vector and the observation vector. Each record inside one of those vectors is linked to an identified template. It is therefore possible to quantify the influence of a given template over the final error. This information is used to retrieve the most contributing templates to the total error. Since the templates are forged using the application name and the severity, reverting this process enables us to identify the application and severity of the templates that contribute most to the errors.

5.2.3.2 Alert Consolidation

```
{"Timestamp": "2022-11-02 10:32:00",                    1
  "Data Centers": [{                                    2
    "Name": "DC-1",                                     3
    "Hosts":[{                                          4
      "Name": "Host-13",                                5
```

```
        "Anomaly Score": 4.5,                                    6
        "Applications": [{                                       7
          "Name": "App-13"                                       8
          "Score Weight": "92.4%"                                9
      }]},                                                       10
      {"Name": "Host-27",                                        11
        "Anomaly Score": 3.2,                                    12
        "Applications": [{                                       13
          "Name": "App-13"                                       14
          "Score Weight": "54.2%"},                              15
        {"Name": "App-9"                                         16
          "Score Weight": "36.8%"}                               17
      ]}]                                                        18
  }]}                                                            19
```

Listing 5.1: An Alert Report

False positive or irrelevant alerting are the main obstacles toward the adoption of an anomaly detection system by the operational teams [117]. At the end of the previous step, for each alert candidate, the system returned the host as well as the top contributing applications. In the alert consolidation step, it is possible to define additional rules to reduce the number of alerts. For instance, alerts linked to similar equipment inside a data center can be grouped, or alerting scenarios can be defined based on the involved applications.

An example of an alert report is provided in Listing 5.1, showing how alert candidates from equipment connected to the same data center are used. For each host, the anomaly scores as well as the weights of the top contributing applications are listed. The ability to generate detailed alert report is a strong point toward Monilog acceptance by operational teams as it gives them detailed information about the potential issue while allowing them to fine-tune the alert settings based on their experience and domain knowledge.

5.3 Classification of Log-Based Anomalies

5.3.1 Within the Litterature

Anomalies encompass various types of events, ranging from breakdowns to performance issues, including security attacks. These anomalies possess distinct characteristics that often necessitate the involvement of different teams, each with their own set of priorities. Effectively classifying a detected anomaly and providing contextual information facilitates the identification of the most appropriate team to address the issue at hand. In our review of the literature, we discovered three studies that explore the automated classification of log-based anomalies.

Zou et al. proposed Uilog [118] (2016), an anomaly classifier that built a fault keyword matrix to classify anomalies. This matrix encodes the probability of a given token to appear in each fault type. This matrix is dynamically learned during the training phase thanks to labeled anomalies. Author's evaluation on classic error affecting CentOS based virtual machines in a cloud illustrates the relevance of their proposal to correlate tokens with specific anomalies.

With WEAC [119] (2017), Pande et al. also exploit token embedded information to detect and classify anomalies by using word2vec techniques. Authors illustrates the superiority of their method comarent to Isolation Forest [88] (IF), and HDBSCAN [120], two traditionnal mahcine learning techniques.

LogClass (2018), by Meng et al., is a classifier trained over log anomalies [121]. Authors applied their proposal to a set of switch logs issued from data centers to detect and classify common errors (e.g., FAN_FAILED, POWER_DOWN, SYSTEM_REBOOT). Logclass approach is based on the bag-of-words technique and learns anomalies labels based on the frequently associated tokens.

A common limitation of these approaches is their reliance on manually established labels to train their classifiers. Acquiring such labeled data is a laborious task, and the resulting trained classifier may not be adaptable to newly detected anomalies. Consequently, these approaches become challenging to maintain, which, in our view, is a key factor contributing to the lack of active research on log-based anomaly classification.

5.3.2 With Monilog

Classifying anomalies can be a subjective task, different organizations may have different ways of handling the same type of events and will assign them different priorities based on their business model. Therefore, instead of trying to group anomalies around known errors, we focused on a way to bring up as much relevant information as possible about an anomaly.

Unlike other approaches, Monilog does not incorporate a classifier module. Instead, when it detects a potential anomaly, it generates a comprehensive report containing relevant information. As depicted in Section 5.2.3, this report includes the concerned equipment, the applications responsible for the anomalies alongside the criticity of the logs linked to those applications. The proportional contribution to the error score for each log template (appname + criticity) is also provided.

Organizations can easily create a tailored anomaly handling process on the Monilog platform by leveraging business rules and expert knowledge. An example at cloud scale could be for instance to group anomalies occurring at the same time if they concern equipment inside a given subnet/data center. One could also use the involved applications to create error handling scenarios based on identified error templates. We believe the flexibility offer by Monilog to be a key factor for his adoption by industrial practitioners in production.

5.4 Perspectives

In this chapter, we presented Monilog, an automated system for detecting anomalies based on log data. Monilog is specifically designed to address the challenges posed by cloud computing platforms, which often consist of numerous complex components of diverse types and undergo frequent updates reflected in log statements. One of the key strengths of Monilog is its capability to generate comprehensive statements that provide detailed information about the detected anomalies and the specific components involved. This inherent flexibility offered by Monilog has been instrumental in its adoption by industrial practitioners operating in real-world production environments.

We were able to implement an instance of Monilog to monitor the European region of 3DS OUTSCALE. In Chapter 6, we illustrate the practical utilities of Monilog by applying it to the detection of Kernel Virtual Machines crashes. Our proposal is quite flexible and many improvements can still be made. We share our thoughts and perspectives on this in Section 7.2 at the end of this manuscript.

CHAPTER 6

Forecasting KVM Crashes Using Monilog

Contents

6.1 Introduction

To assess the practical applicability of our proposal, we implemented a version of Monilog using the logging infrastructure provided by 3DS OUTSCALE and tested it at a cloud region scale. The implemented architecture was running three different models and three different error metrics detailed below. It allows us to validate the real-time performances of our approach as well as its relevance to forecast anomalous events and provide useful insight regarding the anomalous periods.

In this section, we provide a comprehensive overview of the logging environment, collection mode, experimental context, selected models, and metrics. We also discuss their performance in forecasting anomalous events. We have focused our assessment work around two research questions:

- ***RQ3.*** *What is the precision of each of the considered model/metric pairs for the detection of abnormal events ?*
- ***RQ4.*** *Can the considered model/metric pairs forecast the major events that impacted the monitored systems during the experimental period ?*

In Section 6.3.1, we share and comment the result of a precision evaluation of the tested model/metric couples. We used manually generated labels regarding the relevance of the identified anomalous candidates. The section 6.3.3 is dedicated to *RQ4*, as we explore the capability of our proposal to forecast three critical anomalous events that occurred during the experimental period.

6.2 Experimental Setup

6.2.1 Dataset

A Kernel Based Virtual Machine or KVM refers to a Linux system turned into an hypervisor that allows a host machine to run multiple isolated virtual environments, called virtual machines.

A KVM exposes four types of resources to its hosted virtual machines: Computing resources, memory, network access and storage access. Server hard failures are rare events but they can have a negative impact on the hosted client virtual machines.

During 11 consecutive days, our Monilog implementation was reading a Kafka stream filled by syslog messages issued from all the Kernel Based Virtual Machines (KVMs) linked to the 3DS OUTSCALE European cloud region. Several billion lines of logs were consumed during the monitored period at the average rate of 238 logs per host per minute.

6.2.2 Anomalies

During the monitored period, operational teams reported and timestamped three severe crashes. However, during the same period, Monilog alerting system generates a higher number of alert candidates. To evaluate the relevance of Monilog architecture as well as the performance of each model/metric combination, we ran a manual inspection of all the reported abnormal events to assign them one of the following label: *Probable Anomaly* (PA) for the legitimate abnormal events; *False Positive* (FP) for the identified events that are not likely to represent anomalies, and *Unknown* for the remaining ones.

We assign to the *Probable Anomaly* category events such as a high concentration of kernel errors, dropped log messages due to excessive rates or network-related errors. Unusual but not abnormal events such as a large number of virtual machines being deployed simultaneously, booting operations, or admin operations such as network configurations changes were labeled as *False Positive*. Events inside the *Unknown* are linked to log events referring to external systems or metrics. Assigning a clear label to such events is a difficult task as it requires to correlate the information from multiple systems. Given the significant number of such events in our case, we made the decision to set them aside, as inspecting each one in detail was not practically feasible.

6.2.3 Models

The cloud computing logging environment is inherently chaotic, significantly influenced by various client operations. Some of these operations are periodic and run by scripts like the backup or the cleaning of data where other ones are punctual (e.g., manually triggered API calls). It is common for equipment, such as servers, to transition rapidly from an inactive state to a fully loaded state. To address this dynamic nature, forecasting models are utilized to predict the subsequent log count vector based on the previous vectors. In our experimental evaluation, we have chosen and implemented three distinct types of models for this purpose.

We set $r = 24$, the number of last count vectors forwarded to the model. As each vector represents a five-minute time window, this is the equivalent of asking the model to predict the next five minutes based on the last two hours of logs.

6.2.3.1 Repetitor

This model always predicts the most recent count vector as the next output. With this model:

$$\hat{V}(t) = V(t) \tag{6.1}$$

The repetitor model offers a rather simplistic approach to forecasting log traffic, assuming its consistency over time. However, it presents some interesting aspects. As it only considers the last log count vector, it is less influenced by trailing noises. Obtained result displays that coupled with the appropriate error metric, this model can achieve a good precision even if it tends to miss a large number of abnormal events.

6.2.3.2 Mean Channel Value

This model returns the channel wise mean of the last r observations. For this model, $\hat{V}(t)$ is constructed as below:

$$\hat{V}(t)(i) = \frac{\sum_{j=1}^{r} M(j)(i)}{r} \tag{6.2}$$

Another straightforward approach to forecasting log traffic is through the mean channel value model. It aims to mitigate the influence of the chaotic changes such as wide variation affecting one channel by using its over time mean.

6.2.3.3 LSTM Autoencoder

LSTM [122] is a type of Recurrent Neural Network (RNN) that allows the network to retain long-term dependencies between data at a given time from many time steps before. They have been used by a large panel of sequence learning tasks including handwriting recognition, speech recognition, and sentiment analysis. An LSTM-based encoder is used to map an input sequence to a vector representation of fixed dimensionality. The decoder is another LSTM network which uses this vector representation to produce the target sequence.

LSTM autoencoders displayed promising results for multivariate time series forecasting in contexts like supply chain management [123], multi-sensor anomaly detection [124], or spacecraft monitoring [125]. To test the relevance of such models in our cloud computing logging context, we trained an LSTM encoder decoder to reconstruct instances of normal time series. Training log data were collected during the two weeks prior to our 11 days train dataset, they are issued from the same KVM but there is no overlap between the training and the evaluation period. A linear layer on top of the LSTM decoder layer is used to predict the next log count vector $\hat{V}(t)(i)$.

6.2.4 Error Metrics

Accuracy measurement in a forecasting problem is always a subject of debate because of its importance. Workloads can significantly vary between two identical devices and change drastically over time. For instance, a server charge can operate between 0 and 100% of its computing capacity and quickly evolve within this range of values without abnormal. We have chosen three distinct metrics for this experimental evaluation. Each metric is given a forecast vector $\hat{V}$ and an observation vector V and returns a numeric value. We assume here that vectors have the same dimension $|\mathcal{T}| + 1$.

6.2.4.1 Root Mean Square Error

The Root Mean Square Error (RMSE) is a commonly used metric in statistics. It is defined as the square root of the mean square error between the forecast vector $\hat{V}$ and an observation vector V:

$$RMSE = \sqrt{\frac{\sum_{i=1}^{|\mathcal{T}|+1}(\hat{V}(i) - V(i))^2}{|\mathcal{T}| + 1}} \tag{6.3}$$

As it doesn't involve volumetric comparisons, this metric is heavily influenced by the most represented log templates. Two unpredicted logs associated to a rare logging event will have the same weight in the final error than two unpredicted logs associated to a frequent logging event.

6.2.4.2 Symmetric Mean Absolute Percentage Error

The Symmetric Mean Absolute Percentage Error (SMAPE) is also a commonly used metric in statistics, it expresses the accuracy as a ratio defined by the following formula:

$$SMAPE = \frac{100}{|\mathcal{T}| + 1} \sum_{i=1}^{|\mathcal{T}|+1} \frac{|\hat{V}(i) - V(i)|}{|V(i)| + |\hat{V}(i)|} \tag{6.4}$$

It is the absolute difference between an observation V and the forecast $\hat{V}$ divided by half the sum of absolute values of the observation and the forecast. The value of this calculation is summed for every fitted point and divided again by the number of fitted points $|\mathcal{T}|+1$. The lower the SMAPE value of a forecast, the higher is the model accuracy. This error metric involves volumetric comparisons between the expected and obtained number of each logging template. With this metric, rare unpredicted events will have a stronger impact on the final value and fluctuation of the number of common events will be less impacting. One limitation to SMAPE is that if the actual value or forecast value is 0, the value of error will approach for the concerned channel 100%.

6.2.4.3 Log Accuracy Ratio

Mean Absolute Percentage Error metric systematically promotes methods with lower predictions. The Log Accuracy Ratio (LAR) was introduced by Tofallis in 2015 [126] to avoid this bias. LAR is defined as the square logarithm of the ratio observation over the forecast:

$$LAR = \frac{1}{|\mathcal{T}| + 1} \sum_{i=1}^{|\mathcal{T}|+1} ln(\frac{V(i)}{\hat{V}(i)})^2 \tag{6.5}$$

Its properties are similar to SMAPE ones, as it also induces volumetric comparison between the forecasted and the actual values.

6.3 Evaluation

6.3.1 Precision Evaluation

In order to be accepted by technical teams, it is more important for an alerting model to be precise than to be accurate. A significant proportion of false positive can hide the relevant alerts and, in the long run leads the system to be abandoned. Computing the accuracy of each model/metric pair would require knowing all the abnormal events that occurred during the monitored period. As previously discussed, this is impracticable due to the number of logs inside the dataset. However, we spent a consequent amount of time inspecting and labeling all the abnormal sequences raised by the different model/metric pairs. Each sequence was assigned one of the following labels: *Relevant Alert*, *False Positive* or *Unknown*. More details about those labels can be found in Section 6.2.1.

The precision for each model/metric combination is computed based on those labels using the following equation:

$$Precision = \frac{relevant\_alerts}{relevant\_alerts + false\_positives} \tag{6.6}$$

Repetitor			
	RMSE	SMAPE	LAR
# Relevant alerts	281	840	453
# False positive	577	103	98
# Unknown	1 911	2	3
Total	2 769	945	554
Precision	0.327	0.891	0.822
Mean anomaly length in minutes	80.16	17.20	16.26
Anomalous periods time share in % $\times 10^{-3}$	14.6	1.07	0.59

MeanPredictor			
	RMSE	SMAPE	LAR
# Relevant alerts	37	825	352
# False positive	60	497	363
# Unknown	1 274	818	849
Total	1 371	2 140	1 563
Precision	0.381	0.624	0.493
Mean anomaly length in minutes	59.50	54.30	65.32
Anomalous periods time share in % $\times 10^{-3}$	5.37	7.64	6.71

LSTM-AE			
	RMSE	SMAPE	LAR
# Relevant alerts	1	10 458	4 085
# False positive	0	745	865
# Unknown	11 315	2 016	5 842
Total	11 316	13 219	10 792
Precision	N.A	**0.933**	0.825
Mean anomaly length in minutes	5	5	5
Anomalous periods time share in % $\times 10^{-3}$	3.70	4.33	3.53

Table 6.1: Predictor Alerting Precision per Selected Error Metric

It is the ratio of relevant alerts over the total number of alerts. Note, we left aside all the *Unknown* labeled events. As we weren't able to assign them a clear label, we chose to ignore them when computing the precision. Table 6.1 regroups the obtained results for each of the

nine model/metric combinations. Precision for the LSTM-AE/RMSE should be 1.0. Assigning a label to anomalies reported by this pair was a complicated task and, we end up assigning the Unkown label to almost all of them (11 315 over 11 316). We consider this to be a strong sign that reported anomalous period will not be easily exploitable to detect effective anomalies. Regarding this, we set aside the precision result for this combination.

Overall, SMAPE appears to be the precision wise best performing metric, and LSTM-AE/SMAPE the most precise pair. This combination achieves a precision of 0.933 while also being the most verbose one with 13 219 anomaly candidate, 11 203 if we remove the unknown ones. RMSE on its side is the worst performing metric for each model. We explain this by its sensibility to volume change regarding the common templates. Most of the false positives raised by the model/RMSE pairs are linked to a massive deployment of virtual machines, automated CRON jobs or maintenance operations. All of those operations tend to generate a usually high number of log messages. The model/LAR pairs produces in between result, Tofallis [126] concern regarding metrics favoring underforecasting models does not seem to be harmful in our case. Regards to the previously stated results, SMAPE appears to be the go for choice to optimize the precision regarding generated anomaly candidates.

Despite its simplicity, the repetitor model surprisingly achieves high precision. It obtained a precision of 0.891 when coupled with the SMAPE error metric, and a precision of 0.822 when coupled with the LAR error metric. We believe this result to be driven by the chaotic aspect of logs. A repetitor model with a proportional metric such as SMAPE or LAR is good to detect short term major evolution in logging behaviors. Also, the 5-minute duration of the window is huge compared to cloud operations periods that are usually measured in milliseconds. However, the total number of relevant alerts (840 for Repetitor/SMAPE) is way lower than the 10 458 reported by LSTM-AE/SMAPE.

6.3.2 Anomaly Duration

In the lower part of Table 6.1, we indicate the mean anomaly length for the reported anomaly candidates as well as the percentage of the total dataset duration that is flagged as part of an abnormal period (in $\%\times10^{-3}$).

Anomalies reported by LSTM-AE/metric pairs are all of the same duration (5 min), which is also the window length duration. Thus meaning that such model/metric combination never flagged two consecutive periods as abnormal. LSTM-AE forecasting seems to quickly adapt its output to the newly detected logging behaviors. This is a strong point regarding the capability of LSTM-AE neural structure for multivariate time series forecasting as well as for anomaly detection. Still, this fast adaptability seems to be penalizing regarding the discovery of long-term events such as KVM crashes. We discuss this more in the next section when analyzing the ability of model/metric pairs to forecast critical events.

Anomalies reported by MeanPredictor/metric like pairs tend to last around one hour (54.3 minutes for the MeanPredictor/SMAPE pair). This corresponds to almost half of the total duration of past observations forwarded to the model. One explanation is that drastic changes in the logging behavior impact the forecasting output of the model for the next two hours. A positive side of this behavior is that the model is more sensitive to the accumulation of errors over the time.

6.3.3 Critical Events Forecasting

Throughout the 11-day experimental period, the operational teams reported three server crashes. As each crash affected a different KVM and occurred on a different day, we assumed here that they are independent anomalies. Predicting these events could aid the monitoring team in reducing their impact on client-hosted virtual machines. In this section we answer question *RQ4.* by investigating the capacity of the considered model/metric pairs to forecast the reported crashes.

We consigned the obtained results under Table 6.2. For each crash we reported the model/-

metric pairs that detected the crash (i.e., crash time occurs during one of the reported anomaly candidate time periods), and if applicable the forecast. The forecast is computed using the time difference between the reported crash time and the beginning of the associated abnormal period if applicable.

The results indicate that two model/metric pairs detected at least one crash, while only one managed to detect all of them. The MeanPredictor/LAR detects two crashes and the MeanPredictor/SMAPE detects all of the three reported crashes. For the five detected crashes, the forecasting time is superior to 80 minutes. Giving the low number of alleged crashes, we are working with, it is difficult to reach any hard conclusion. However, obtained results support our hypothesis that KVM log files can be used to predict server crashes.

MeanPredictor model was the only one able to forecast some of the reported crashes. We believe this to be linked to this model sensitivity to the accumulation of small anomalies over time. When diving into prior to crash KVMs logging behavior, we observed a drift within the hours before the crash. A higher than the usual number of logs with error or warning severity was reported during this period. MeanPredictor/SMAPE pair appears to be specially efficient regarding the detection as well as the long-time alerting regarding those drifts. Figure 5.2 displays the over time error of the SMAPE error metric over MeanPredictor forecasts. The crash occurred during Day 3, the crash error time is clearly visible as the associate reported error value is the highest for the considered KVM during the experimental period.

Despite their abnormal nature, both the Repetitor and LSTM-AE models swiftly adapt to new behaviors. Repetitor and LSTM-AE models coupled with SMAPE or LAR metrics raised anomaly at the same time as the MeanPredictor/SMAPE and MeanPredictor/LAR models for the alleged crashes. However, they stop raising alert after a short period, around 30 minutes for Repetitor/metric pairs and after one abnormal period (5 min) for LSTM-AE/metric pairs. Regarding this, we could debate about the ability of such models to forecast crashes because to a certain extent they detect their premises. When looking at the alert candidate detailed report raised by repetitor and LSTM-AE models, it was obvious that it was an alleged anomaly

but hard to predict a crash. On the other hand, MeanPredictor generated reports give us more clues.

Repetitor				
		RMSE	SMAPE	LAR
Crash 1	Detected	✗	✗	✗
	Forecast (minutes)	N.A	N.A	N.A
Crash 2	Detected	✗	✗	✗
	Forecast (minutes)	N.A	N.A	N.A
Crash 3	Detected	✗	✗	✗
	Forecast (minutes)	N.A	N.A	N.A

MeanPredictor				
		RMSE	SMAPE	LAR
Crash 1	Detected	✗	✓	✓
	Forecast (minutes)	N.A	**120**	**120**
Crash 2	Detected	✗	✓	✓
	Forecast (minutes)	N.A	**80**	**80**
Crash 3	Detected	✗	✓	✗
	Forecast (minutes)	N.A	**85**	N.A

LSTM-AE				
		RMSE	SMAPE	LAR
Crash 1	Detected	✗	✗	✗
	Forecast (minutes)	N.A	N.A	N.A
Crash 2	Detected	✗	✗	✗
	Forecast (minutes)	N.A	N.A	N.A
Crash 3	Detected	✗	✗	✗
	Forecast (minutes)	N.A	N.A	N.A

Table 6.2: Model/Metric Pairs Ability to Forecast the Major Anomalous Events

6.4 Lesson Learned

In this chapter, we presented Monilog a log-based anomaly detection system designed for the cloud computing context. It is able to detect anomalous events in contexts where log relationships are not identifiable and to generate detailed output regarding the reported events including the concerned system and the applications whose behavior led to raise an alert. We conducted an evaluation of our proposal at industrial scale using 11 consecutive days of logs issued from the KVMs of the european cloud region operated by 3DS OUTSCALE. For this evaluation, we considered three forecasting models: Repetitor, MeanPredictor and LSTM-AE and three error metrics: RMSE, SMAPE, LAR. Considering the accuracy of each model/metric pair, SMAPE emerged as the top-performing error metric, with LSTM-AE proving the most effective model for detecting anomalies. During the monitored period, three KVM crashes were reported, and the MeanPredictor/SMAPE combination was able to predict all of them with at least 80 min of forecasts.

6.5 Perspectives

The results obtained in our study provide strong support for our hypothesis regarding the utility of logs in detecting and forecasting abnormal events. The implementation of our Monilog system demonstrated accurate identification of abnormal periods and offered valuable insights into the associated applications within a KVM environment. Moving forward, we aim to expand the monitoring capabilities of Monilog by conducting experiments on a wider range of cloud computing equipment and software. A key concern of our work is reducing the number of false alerts. As we broaden the monitoring scope, it is expected that abnormal events will generate multiple alert candidates as they become linked to more devices. To address this, we plan to experiment with combining Monilog's reported anomaly candidates with existing key performance indicators (KPIs) and metrics, resulting in fewer alerts with enhanced contextual information. Additionally, we believe that leveraging cloud-specific rules, such as geographic

distribution of devices, can facilitate the combination of anomaly candidates and their associated contexts.

Regarding the labeling of the anomalies in our experimental study, we choose to put admin operations in the *False positive* category, such events can be the sign of malicious events run by an infiltrated agent with admin privileges. As the focus of this work is on the platform and resources monitoring, we chose to set such scenario aside and classified the linked event as normal. Also, the error metric should be picked carefully and be driven by your context [127]. In our cloud computing environment, non-homogeneity is a major concern. Error metric that includes proportional comparisons (SMAPE and LAR) gave us better results than the other one (RMSE). In the future we can imagine crossing information from multiple error metrics to alert on specific scenarios.

CHAPTER 7

Conclusion and Perspectives

Contents

7.1 Conclusion

Cloud computing platforms offer a globally accessible pool of shared and highly available processing and storage resources. These platforms have gained immense popularity due to their cost-effective distribution of computing infrastructure among multiple organizations. However, effectively monitoring the growing number of IT resources on the provider side presents a significant challenge. Logs serve as a valuable resource for various tasks, including inspection, maintenance, development, and alerting.

In this manuscript, we present our research, which aims to address the issue of automated log-based anomaly detection within cloud computing platforms. Our work aligns with the fields of data mining, cloud computing, deep learning, and distributed systems. The key contributions of our research are as follows:

1. Literature Review: We conduct a comprehensive review of existing research in the areas of log parsing and log-based anomaly detection.

2. Robustness and Tuning of Spell and Drain: We investigate the robustness and fine-tuning aspects of Spell and Drain, two state-of-the-art online parsing methods.

3. Novel Parsing Techniques: We propose novel, more robust and efficient approaches for parsing log messages in an online and distributed manner.

4. Automated Anomaly Detection Architecture: We introduce Monilog, an automated architecture designed for detecting anomalies within a cloud infrastructure through log analysis. This architecture leverages recent advancements in deep learning and provides explanations for the flagged events.

5. Cloud-Scale Experimentation: We present a large-scale experimentation of Monilog, an anomaly detection framework, using KVM logs.

Throughout the manuscript, we provide in-depth discussions and analysis for each of the above points, utilizing a combination of open-source and 3DS OUTSCALE log data. We believe

that our work significantly contributes to the advancement of automated log-based anomaly detection in cloud computing platforms and offers valuable insights for practitioners and researchers in this field.

7.1.1 Contributions to the Log-Parsing Field

Labeling logs in complex environments can be a laborious task. For an automated log-based anomaly detection system, it is crucial to have a log parsing solution that is both robust to log changes and adjustable without relying on labeled data. In Chapter 3 of our study, we delve into the impact of hyperparameter tuning on Spell and Drain, two widely used log parsing methods. Our investigation reveals that there are no universally applicable values for these hyperparameters, emphasizing the necessity for more robust parsing solutions.

In Chapter 4, we introduce USTEP, an online log parsing approach based on an evolving tree structure, able to discover and encodes new parsing rules on the run. The comparative complexity analysis highlights the fact that USTEP is the only existing method to achieve constant time complexity as it is not influenced by the state of its memory structure. Experimental results on a panel of 14 datasets including one industrial dataset issued from 3DS OUTSCALE logging infrastructure show the superior parsing effectiveness (+3.4% at 93.7%) and robustness of USTEP when compared to other state-of-the-art techniques.

Log parsing is a mandatory first step for many log mining methods. Therefore, parsing time can become a bottleneck for solutions requiring real-time processing in log intensive environments. In those cases, centralized log parsing fails to produce on the fly output. The USTEP-UP framework introduced in Chapter 3 helps to run multiple USTEP instances in parallel while conserving parsing consistency.

7.1.2 Automated Log-Based Anomaly Detection in Cloud Environment

In Chapter 5 of our research, we introduce Monilog, an anomaly detection system specifically designed for the cloud computing context. Monilog is capable of detecting anomalous events in

environments where the relationships among logs are not easily identifiable. Monilog utilizes a model to forecast the log traffic and error metrics, which enables the evaluation of its forecasting accuracy. An abnormal period is identified when the associated error reaches an unusual value, indicating potential anomalies. One of the unique aspects of Monilog lies in its parsing and embedding of log events. This approach enables the system to generate comprehensive summaries pertaining to the reported anomaly candidates, including details about the systems and applications involved. By leveraging these capabilities, Monilog enhances log-based anomaly detection in the cloud computing environment, offering valuable insights for identifying and addressing irregular events that may occur within complex log structures.

In Chapter 6, we conducted a comprehensive evaluation of our proposal at an industrial scale using 11 consecutive days of logs obtained from the KVMs in the European cloud region operated by 3DS OUTSCALE. The evaluation aimed to assess the effectiveness of our approach in raising relevant alerts for anomaly detection. During the evaluation, we considered three different forecasting models: Repetitor, MeanPredictor, and LSTM-AE, along with three error metrics: RMSE, SMAPE, and LAR. Our primary focus was to evaluate the performance of these nine model/metric pairs in identifying and alerting abnormal periods. Based on the results, the SMAPE error metric emerged as the best performing metric, exhibiting the highest efficacy in detecting abnormal periods. Additionally, the LSTM-AE model demonstrated the highest effectiveness in detecting such abnormal periods. Throughout the monitored period, three instances of KVM crashes were reported. Notably, the MeanPredictor model successfully predicted all of these crashes, providing forecasts at least 80 minutes prior to each event. This outcome serves as a validation of the effectiveness and relevance of Monilog in forecasting abnormal events within cloud computing infrastructure. The evaluation conducted on real-world data from the European cloud region underscores the practical utility and reliability of our proposed approach, reinforcing its potential for enhancing anomaly detection and forecasting within cloud computing environments.

7.2 Perspectives

7.2.1 Data Mining

Our research is dedicated to the identification of anomalies within cloud computing platforms, specifically focusing on the analysis of log data. Within this context, we have developed USTEP, a log parsing solution that we propose, along with its distributed version USTEP-UP, which provides a more robust and efficient approach to log parsing.

Although significant progress has been made, parsing errors remains a concern. It is important to acknowledge that no parsing solution can achieve flawless parsing in all logging contexts. While USTEP has exhibited exceptional performance, we believe there is scope for improvement. Currently, it lacks the ability to group together logs originating from the same log statement but containing a different number of tokens.

To address this limitation, we propose exploring the incorporation of virtual links between log templates using techniques such as the Longest Common Subsequence (LCS) algorithm. Although this enhancement may marginally slow down the parsing process, it has the potential to enhance the accuracy of log grouping.

The extraction of data from logs is a critical step for log-based applications in general. The most precise information about the current state of a system is often embedded within the log message and can only be accessed through parsing. Currently, little utilization is made of non-numerical variables, as existing approaches primarily focus on template mining. This limitation is associated with the diverse nature of the embedded variables, including text, IP addresses, URLs, numbers, hashes, and more. Exploiting this information can be pivotal for numerous log-based applications in domains such as monitoring, security, and usage mining.

Furthermore, we posit that log parsers can be employed for a broader range of data mining tasks beyond log parsing itself. They can prove invaluable in detecting patterns in streams of messages or extracting underlying information from textual content. Through two experiments conducted at a recreational scale, we have demonstrated the intriguing potential of USTEP in extracting content from tickets within the JIRA work management software or from invoices.

7.2.2 Log-Based Anomaly Detection

The experimental results conducted on a cloud platform scale scale demonstrate the promising potential of Monilog in achieving automated log-based anomaly detection. Further exploration of Monilog's capabilities should focus on multi-component anomaly detection. While our practical analysis solely utilized logs from the KVM, it would be interesting to extend monitoring to a broader range of logging sources, such as storage devices and cloud hypervisors. The range of abnormal events that Monilog can detect is extensive, with our current focus being on virtualization-induced server crashes. However, this detection capability could be expanded to include security events or any other form of unusual logging behavior.

On a more technical note, advancements in the field of machine learning can greatly benefit Monilog by providing opportunities to experiment with new log forecasting models. Conducting experiments with different model/metric combinations would enhance our understanding of how to optimize the Monilog architecture specific contexts.

Regarding the embedding step of Monilog, we deliberately opted not to employ log parsing. This decision was driven by the desire to assess the potential of Monilog using the simplest approach as a benchmark. However, in the future, we believe that incorporating log parsing would be advantageous, as it would enable the utilization of a wider range of information for anomaly detection and the generation of anomaly reports. This, however, would require dealing with parsing errors.

Monilog is designed to detect concept drifts in system logging behavior and provide detailed outputs related to such drifts. Nevertheless, this approach can also be adapted to capture various events. For instance, we envision Monilog being applied to track the propagation of software updates within a cloud platform or to analyze usage patterns across different system components.

7.2.3 Safety and Monitoring

By carefully examining logs, security professionals can identify potential security breaches, malicious activities, and unauthorized access attempts. Logs enable the detection of suspicious patterns, anomalies, and indicators of compromise, facilitating early warning and timely incident response. Additionally, logs play a crucial role in forensic investigations, providing a detailed record of events and actions that can be analyzed to reconstruct timelines and identify the root cause of security incidents. Security use cases usually required fine-grained analysis as attackers take care of leaving the smallest number of traces.

Security use cases typically necessitate fine-grained analysis, as attackers are adept at minimizing their traceable activities. A challenge extensively addressed in the literature is that the majority of log analyses are conducted post-mortem [128]. Consequently, such approaches are unable to promptly address ongoing attacks. In our future research, we aim to explore automated methodologies for swiftly detecting security attacks by leveraging multiple data sources, including logs.

Correlating log data with other data sources such as KPIs, metrics or architectural information could help refine the analysis of abnormal events, reduce the number of false positive, detect weaker signals or enrich the anomalous context. We believe that working with heterogenous sources is key to develop better monitoring systems able to detect a wider range of abnormal events including malicious and hardly traceable events.

This is challenging as working with a single data sources can already be a big data challenge as this was discuss a lot inside this manuscript. Combining multiple sources can therefore pose an even bigger data challenge [129]. Still we believe this research field to be the next big step in building intelligent monitoring system for complex platforms such as cloud ones.

CHAPTER 8

Extended Summary in French

Contents

8.1 Introduction

8.1.1 Contexte

La flexibilité et la haute disponibilité des ressources informatiques mise à disposition par les plateformes de cloud computing stimulent une adoption massive du modèle. Le secteur jouit d'une croissance toujours dynamique avec une prévision de 29.8% pour le secteur public en 2023 [4]. Du côté des fournisseurs, le maintien en activité et le développement d'un parc d'équipements et de services en croissance constante posent des contraintes d'automatisation.

Dans le cadre de cette thèse, nous étudions les registres machines (logs) comme source d'information pour la création d'un système innovant et autonome de détection des anomalies dans les plateformes de cloud computing. Nos travaux ont été réalisés dans le cadre d'un partenariat industriel avec 3DS OUTSCALE, fournisseur français cloud souverain. Cette collaboration nous a permis d'échanger avec des experts techniques du domaine, de développer une meilleure connaissance des enjeux et des limitations des systèmes existants et de réaliser des expériences à l'échelle sur des données réelles.

8.1.2 Contributions

Nous avons organisé nos travaux de recherches autour de 3 axes:

8.1.2.1 Axe 1: Structuration des Logs

Les logs informatiques sont des données semi-structurées. Chaque log a un entête qui suit un format défini par convention, incluant des champs tels que la date et l'heure, le système, l'application concernée, ou la criticité. Le format des messages de logs lui, est libre et laissé à la discrétion des développeurs. Cette flexibilité a permis une adoption massive des logs en pratique, mais est également un frein à leur exploitation automatique. Afin de récupérer les données relatives à l'état courant du système il nécessaire de pouvoir les extraire du message.

Dans cet axe, nous explorons les méthodes existantes de structuration des messages logs et nous nous intéressons à l'optimisation de leurs hyper paramètres afin d'identifier des valeurs

génériques. Les résultats de notre étude mettant en limite la non-existence de telles valeurs nous ont poussés à proposer USTEP une nouvelle méthode de structuration des messages logs, plus robuste aux différents environnements et présentant une meilleure complexité temporelle que l'état de l'art. Le besoin de temps réels étant un prérequis fort pour un système de détection d'anomalies, nous avons également introduit USTEP-UP une architecture pour faire tourner plusieurs instances d'USTEP en parallèle et permettre le passage à l'échelle.

8.1.2.2 Axe 2: Détection des Anomalies dans des Infrastructures Cloud

Les logs sont déjà activement utilisés pour la détection d'anomalies et depuis 2017, de nombreuses méthodes basées sur des approches de deep learning ont été publiées [24; 29; 71]. Ces méthodes présentent des précisions élevées, cependant elles ont besoin de pouvoir regrouper les logs en séquences afin de détecter des anomalies. En pratique, à l'échelle d'une plateforme de cloud computing, ce prérequis est pratiquement impossible à obtenir à cause de la forte volumétrie, de la multiplicité des sources de logs et de l'absence d'identifiants uniques de transactions dans de nombreuses sources.

Nous avons proposé Monilog, un système autonome de détection des anomalies adapté aux besoins de robustesse et de rapidité des plateformes cloud et qui ne nécessite pas de devoir lier les logs entre eux. Monilog est inspiré du domaine des séries temporelles multivariées, il prédit l'activité log d'une machine à l'aide d'un modèle et se sert de son erreur de prédiction pour détecter des comportements anormaux. Notre implémentation de Monilog à des fins de surveillance de la région Europe de 3DS OUTSCALE nous a permis de valider son efficacité pour la prédiction de comportements anormaux liés aux serveurs de virtualisation (KVM).

8.1.2.3 Axe 3: Caractérisation des Anomalies

Lorsqu'une anomalie est détectée, l'information remontée à une équipe humaine doit indiquer le plus précisément possible la cause et le contexte du problème. Pour assurer une résolution efficace, l'identification du contexte est aussi importante que l'anomalie elle-même afin de cibler les experts les plus qualifiés pour la gérer. Les travaux liés à cet axe comprennent une revue

de la littérature des systèmes de classification existants pour les anomalies basées sur les logs et leur pertinence dans un contexte de cloud computing.

En raison de l'évolution constante affectant le comportement des composants sous-jacents d'une plateforme cloud, la classification des anomalies est une tâche fastidieuse et non pertinente sur le long terme. Aussi, les professionnels peuvent avoir différentes manières de traiter les mêmes anomalies en fonction de leur contexte. Ces préoccupations nous ont poussés à proposer et à intégrer dans Monilog la possibilité d'identifier le contexte des erreurs remontées. Ce contexte inclut les équipements et applications concernés ainsi qu'une évaluation de sa criticité. Ces rapports permettent aux praticiens de créer leurs propres règles métier, plus flexibles et pertinentes qu'un classificateur.

8.2 État de l'Art

8.2.1 Détection des Anomalies à l'aide de Logs

La recherche de mots clés ou de motifs est l'utilisation la plus simple qui peut-être faite des logs pour détecter des anomalies. Bien qu'efficaces pour des scénarios bien connus et identifiés, ce type d'approches ne sont pas capables de découvrir de nouveau scénario et dépendent du travail d'experts pour être enrichie et mises à jour.

Ce besoin d'automatisation des processus a servi de motivation à de nombreux travaux d'exploitation des logs. Durant les deux dernières décennies, de multiples approches de détection automatisée d'anomalie à partir de logs ont été proposées, principalement basées sur des méthodes d'apprentissage automatique ou d'apprentissage profond.

D'après Bhanage et al. [36], plus de 87 publications portant sur des méthodes de détection d'anomalies à partir de logs ont été publiées entre 2016 et 2021. Nous avons regroupé dans la Table 8.1 les méthodes les plus représentatives de l'état de l'art avec leurs spécificités.

Année	Papier	Méthode	Année	Papier	Méthode
2007	[64]	SVM	2019	[29]	Bi-LSTM
2009	[65]	PCA	2020	[66]	IF + Autocencoder
2012	[67]	IM	2020	[68]	BERT + Bi-LSTM
2016	[69]	Clustering	2021	[70]	Transformer
2017	[71]	LSTM	2021	[72]	Clustering + CNN
2017	[73]	Clustering	2021	[74]	Transformer
2018	[75]	CNN	2021	[76]	GAN
2018	[77]	Clustering	2021	[78]	Adversarial Network
2019	[79]	Transformer	2021	[80]	Transformer
2019	[24]	Bi-LSTM	2022	[81]	TCN

Table 8.1: Méthodes de Détection Automatisée des Anomalies, un Aperçu

8.2.1.1 Méthodes Traditionnelles

À notre connaissance, la première méthode de détection d'anomalies basée sur des logs a été évoquée par Liang et al. en 2007 [64]. Le système proposé est basé sur l'ensemble de classificateurs SVM (Support Vector Machines) et vise à détecter les défaillances affectant le supercalculateur IBM BlueGene/L. Dans la catégorie des méthodes traditionnelles, on retrouvera par la suite des approches basées sur d'autres méthodes d'apprentissage automatique telles que Principal Component Analysis [65], Invariant Mining [67] ou du partitionnement [69; 73].

8.2.1.2 Méthodes Basées sur de L'apprentissage Profond

Présentée en 2017, Deeplog [71] est la première méthode proposée basée sur des réseaux de neurones. Elle utilise un réseau de neurones de type LSTM pour détecter des anomalies dans des séquences de logs. Deeplog a rapidement été étendu par des chercheurs de Microsoft [24] et de Huawei [29]. Les deux équipes ont utilisé une structure LSTM bidirectionnelle et ajouté des mécanismes de résilience face aux évolutions des comportements logs des systèmes ciblés.

Depuis, de nombreuses approches d'apprentissage profond ont été proposées, les méthodes utilisées suivant les tendances et les avancées du domaine. Ces méthodes partagent cependant un mécanisme commun, ils dépendent de solution de structuration des logs pour extraire les patrons sous-jacents et autres informations utiles des messages de logs.

8.2.2 Structuration des Logs

Les approches de structuration des logs visent à séparer les parties injectées du patron d'un message afin de reconstituer la ligne de code qui l'a générée. Elles permettent ainsi de regrouper ensemble des logs tu mêmes types et d'accéder à des informations intégrées aux messages. Les avancées dans le domaine de la structuration de logs ont un impact direct les performances des méthodes de détection automatisée des anomalies.

Les méthodes existantes peuvent être regroupées en deux familles. Les approches permettant de traiter des messages en flux, et celles qui ont besoin d'effectuer plusieurs passes sur des jeux de données. Dans notre contexte, les seules méthodes pertinentes tombent dans la première catégorie, les logs étant générés en flux.

À notre connaissance, 4 méthodes de structuration des logs en flux existent dans la littérature: SHISO [100] et LenMa [101] basées sur des méthodes de partitionnement, Spell [102] qui utilise l'algorithme de plus grande séquence commune; et Drain [103; 104] une approche basée sur un arbre de recherche. Parmi elles, Drain est souvent citée comme la méthode la plus précise et recommandée par plusieurs études [1; 29; 105].

8.3 Étude de Méthodes Robustes pour la Structuration des Logs

Les méthodes de structuration des logs étant utilisées en amont des méthodes de détection d'anomalies, leur précision peut avoir une influence directe sur les méthodes en aval. Une de nos études sur Deeplog couplé avec Drain montre qu'une baisse 20% de la précision de structuration peut entrainer une baisse de 70% de la précision de détection des anomalies [31].

8.3.1 Impact des valeurs des Hyperparamètres sur les Méthodes de Référence

Les principaux facteurs influant sur la précision d'une méthode de structuration sont les valeurs de ses hyperparamètres et les expressions régulières qui lui sont fournies pour prétraiter les messages. Isoler un jeu de données représentatif d'un environnement cloud et le labéliser pour paramétrer les méthodes est une tâche ardue et non pérenne dans le temps.

Nous nous sommes penchés sur l'existence de valeurs par défaut permettant des performances stables sur différents jeux de données pour Spell et Drain. Notre étude met en lumière l'inexistence de telles valeurs ainsi que les performances fluctuantes en termes de précision maximale obtenue sur des logs issus de systèmes différents [32]. Ces résultats nous ont motivés à explorer et proposer USTEP, une méthode de structuration des logs plus précise et plus robuste.

8.3.2 USTEP, une Méthode de Structuration plus Robuste et plus Rapide

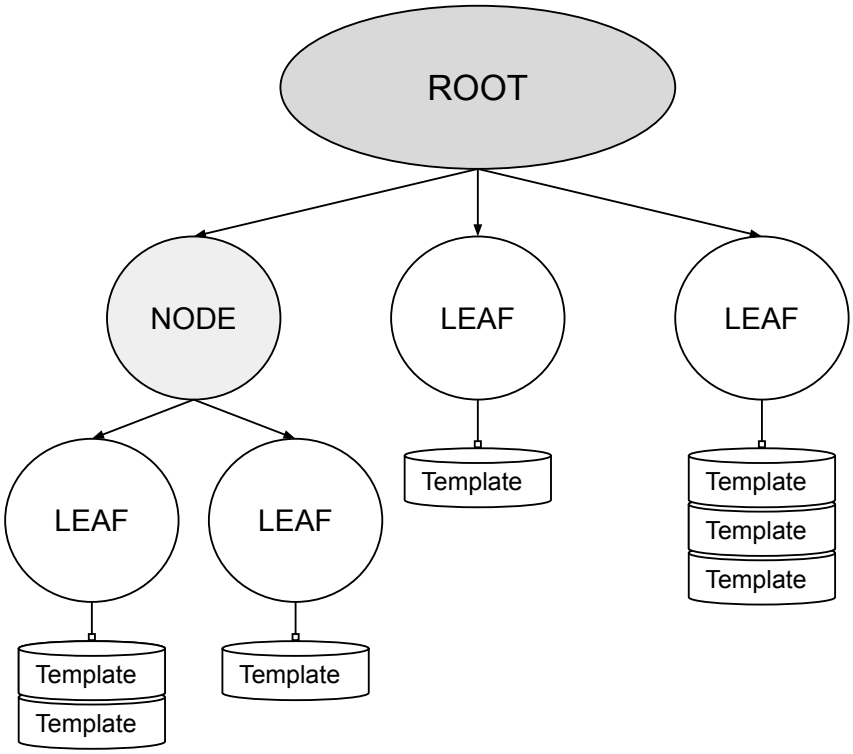


Figure 8.1: USTEP Structurelle Mémorielle Arborescente

Basée sur une structure arborescente qui évolue au fil des messages traités, USTEP est capable traiter en flux des messages de logs. Sa structure mémorielle est basée sur 4 types d'éléments (Figure 8.1): 1/ une racine qui sert de point d'entrée aux actions de recherche; 2/ des noeuds internes qui encodent les règles de parcours de l'arbre; 3/ des feuilles qui regroupent les patrons; et 4/ les patrons déjà découverts.

Les messages entrent par la racine et suivent les règles des noeuds internes jusqu'à rejoindre une feuille. Ils sont alors comparés aux patrons déjà découverts pour savoir s'ils peuvent y être assimilés. Le cas échéant, un nouveau patron est créé à partir du message courant. Quand une feuille contient trop de patrons, elle divise, se transformant en un noeud interne relié à plusieurs nouvelles feuilles. Cette étape d'éclatement permet de garder un temps de traitement rapide tout en créant des règles de descentes spécifiques aux logs rencontrés.

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**
Big-comp	0.480	-	-	-	**0.816**

Table 8.2: Précision des Méthodes de Structuration

Notre évaluation d'USTEP [31] sur 11 jeux de données issus de différents systèmes réels nous a permis de mettre en valeur la supériorité de notre méthode en termes de précision (+3.34% par rapport à Drain) et de robustesse avec un écart type deux fois plus faible que Drain. USTEP est également la seule méthode obtenant une complexité temporelle constante en pire cas.

Afin de pouvoir s'adapter aux changements de charge, nous proposons USTEP-UP, une architecture permettant de faire fonctionner plusieurs instances d'USTEP de manière distribué, sans partage de mémoire, mais avec une consistance de structuration [31].

8.4 Proposition de Monilog un Système Innovant de Détection d'Anomalies à Partir de Logs

Les limites que nous avons identifiées dans notre état de l'art des méthodes de détection des anomalies sont: 1 une sensibilité aux erreurs de structuration; 2 le besoin de pouvoir regrouper les logs en séquences. Le point 2 étant non atteignable de manière réaliste dans le cadre d'une plateforme cloud, de par la diversité des environnements de logs impliqués et leur évolution, nous avons proposé Monilog, une méthode de détection d'anomalie ne nécessitant pas ce prérequis. Monilog est également capable de générer des rapports sur les erreurs remontées incluant le système et les applications concernées.

8.4.1 Architecture de Monilog

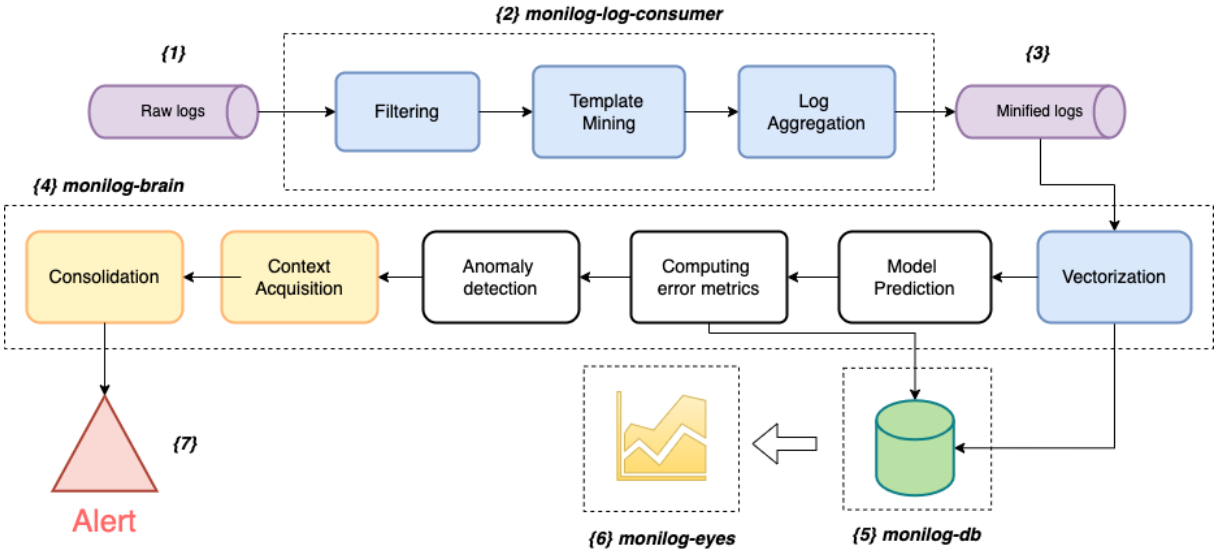


Figure 8.2: Architecture et Fonctionnement de Monilog

L'architecture de Monilog (Figure 8.2), est divisée en trois parties.

1. En bleu, Monilog structure les logs et les regroupe par patrons et par fenêtres temporelles. Dans les phases de vectorisation, le patron assigné à chaque log est constitué de la combinaison de son application et de sa criticité. Nous avons fait le choix ici de ne pas dépendre

de solution de structuration des logs afin d'éliminer les erreurs potentielles.

2. En blanc, il cherche des événements anormaux en comparant ses prévisions d'activité avec la réalité. Les prévisions sont faites par des modèles recevant en entrée les derniers vecteurs liés à un équipement. Les prédictions sont ensuite comparées aux vraies observations pour la même période grâce à des métriques d'erreurs. L'écart par rapport à l'erreur moyenne sur une période concernée sert à déterminer l'anormalité. La Figure 8.3 est un exemple de série temporelle avec, en vert l'erreur moyenne et en rouge le seuil d'alerte.

3. Finalement, en jaune il crée des rapports d'incidents à partir des anormalités détectées. L'utilisation de l'application est de la criticité du message pour créer le patron d'un template permet également d'inverser le lien est de faire ressortir les applications les plus contributrices à l'anormalité.

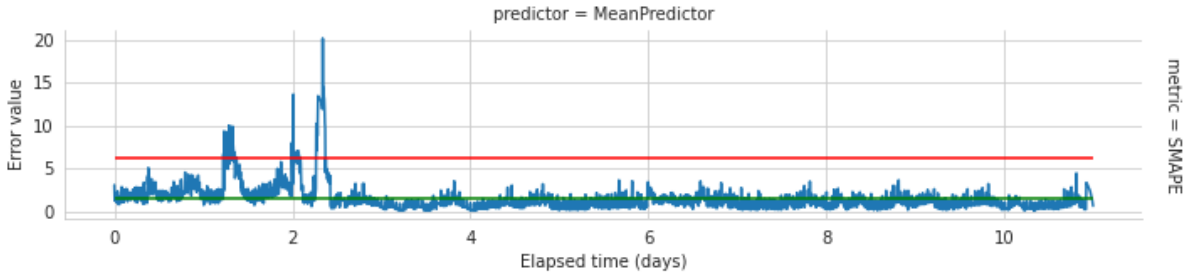


Figure 8.3: Un Exemple de Série Temporelle d'Erreurs

8.4.2 Utilisation de Monilog pour la Surveillance de KVM

Afin de tester le potentiel de Monilog pour le monitoring d'une plateforme cloud, nous avons utilisé les logs générés par les serveurs de virtualisation (KVM) d'un centre de données de 3DS OUTSCALE pendant 11 jours. Dans notre étude, nous évaluons trois modèles de prédiction (Repetitor, MeanPredictor, LSTM-AE) et trois métriques d'erreur (RMSE, SMAPE, LAR) pour un total de 9 paires modèle/métrique.

Les résultats obtenus (Table 8.3) mettent en avant la SMAPE comme la métrique permettant d'obtenir la meilleure précision sur tous les jeux de données. Et le LSTM-AE comme le

Repetitor			
	RMSE	SMAPE	LAR
# Alertes vérifiées	281	840	453
# Faux Positifs	577	103	98
# Inconnu	1 911	2	3
Total	2 769	945	554
Precision	0.327	0.891	0.822
MeanPredictor			
	RMSE	SMAPE	LAR
# Alertes vérifiées	37	825	352
# Faux Positifs	60	497	363
# Unknown	1 274	818	849
Total	1 371	2 140	1563
Precision	0.381	0.624	0.493
LSTM-AE			
	RMSE	SMAPE	LAR
# Alertes vérifiées	1	10 458	4 085
# Faux Positifs	0	745	865
# Inconnu	11 315	2 016	5 842
Total	11 316	13 219	10 792
Precision	N.A	**0.933**	0.825

Table 8.3: Précision des Paires Modèle/Métrique

modèle plus précis, quand couplé avec la SMAPE. Nous avons inspecté manuellement tous les événements remontés, cependant nous n'avons pas toujours pu établir un diagnostic franc. Nous avons donc fait le choix d'introduire un label Inconnu pour ces événements.

Durant la période d'évaluation, trois crashs de KVM ont été remontés par les équipes. Pour ces événements majeurs, seule les combinaisons MeanPredictor/SMAPE et MeanPredictor/LAR ont pu prédire les événements. Cependant, dans le cas de MeanPredictor/SMAPE la prédiction est chaque fois 80 minutes avant occurrence.

8.5 Conclusion Générale

Dans ce manuscrit, nous présentons nos travaux de recherches visant à proposer un système autonome et innovant de détection des anomalies dans les infrastructures de cloud computing en exploitant les logs. Nos contributions sont les suivantes:

- Un état de l'art du domaine de la détection d'anomalies à partir de logs et du domaine de structuration des logs.
- Une étude de la robustesse de Spell et Drain ainsi que de leur paramétrage automatique.
- Nous avons proposé USTEP, un algorithme de structuration des logs et USTEP-UP sa version distribuée.
- Nous avons introduit une nouvelle architecture pour la détection automatique d'anomalie à partir de message logs, capable de générer des rapports sur les anomalies détectées.
- Une évaluation à l'échelle d'un cloud de notre proposition en exploitant des logs issus de serveurs de virtualisation.

Tout au long du manuscrit, nous fournissons des discussions et des analyses détaillées pour chacun des points susmentionnés, basées une combinaison de données de journal open source et 3DS OUTSCALE. Nous pensons que notre travail contribue à l'avancement de la détection automatisée des anomalies basée sur les logs dans les plateformes de cloud computing et fournit des informations précieuses aux praticiens et aux chercheurs dans ce domaine.

References

[1] J. Zhu, S. He, J. Liu, P. He, Q. Xie, Z. Zheng, and M. R. Lyu, "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, 2019. xii, 8, 21, 22, 23, 28, 30, 32, 35, 46, 47, 98

[2] T. Dillon, C. Wu, and E. Chang, "Cloud computing: issues and challenges," in *2010 24th IEEE international conference on advanced information networking and applications*, pp. 27–33, Ieee, 2010. 2

[3] W. Kim, "Cloud computing: Today and tomorrow.," *J. Object Technol.*, vol. 8, no. 1, pp. 65–72, 2009. 2

[4] "Gartner public cloud sector estimated growth for 2022." `https://www.gartner.com/en/newsroom/press-releases/2022-10-31-gartner-forecasts-worldwide-public-cloud-end-user-spending-to-reach-nearly-600-billion-in-2023&`. Accessed: 2023-03-27. 2, 94

[5] C. Guo, L. Yuan, D. Xiang, Y. Dang, R. Huang, D. Maltz, Z. Liu, V. Wang, B. Pang, H. Chen, *et al.*, "Pingmesh: A large-scale system for data center network latency mea-

surement and analysis," in *Proceedings of the 2015 ACM Conference on Special Interest Group on Data Communication*, pp. 139–152, 2015. 2

[6] L. Decker, D. Leite, L. Giommi, and D. Bonacorsi, "Real-time anomaly detection in data centers for log-based predictive maintenance using an evolving fuzzy-rule-based approach," in *2020 IEEE International Conference on Fuzzy Systems (FUZZ-IEEE)*, pp. 1–8, IEEE, 2020. 2

[7] W. Wang, X. Du, D. Shan, R. Qin, and N. Wang, "Cloud intrusion detection method based on stacked contractive auto-encoder and support vector machine," *IEEE transactions on cloud computing*, vol. 10, no. 3, pp. 1634–1646, 2020. 2

[8] H. Liang, L. Song, J. Wang, L. Guo, X. Li, and J. Liang, "Robust unsupervised anomaly detection via multi-time scale dcgans with forgetting mechanism for industrial multivariate time series," *Neurocomputing*, 2020. 2

[9] P. Liu, Y. Chen, X. Nie, J. Zhu, S. Zhang, K. Sui, M. Zhang, and D. Pei, "Fluxrank: A widely-deployable framework to automatically localizing root cause machines for software service failure mitigation," in *2019 IEEE 30th International Symposium on Software Reliability Engineering (ISSRE)*, pp. 35–46, IEEE, 2019. 2

[10] Y. Wang, Z. Wang, Z. Xie, N. Zhao, J. Chen, W. Zhang, K. Sui, and D. Pei, "Practical and white-box anomaly detection through unsupervised and active learning," in *2020 29th International Conference on Computer Communications and Networks (ICCCN)*, pp. 1–9, IEEE, 2020. 2

[11] A. Das, F. Mueller, C. Siegel, and A. Vishnu, "Desh: deep learning for system health prediction of lead times to failure in hpc," in *Proceedings of the 27th International Symposium on High-Performance Parallel and Distributed Computing*, pp. 40–51, 2018. 2

[12] S. Zhang, Y. Liu, W. Meng, Z. Luo, J. Bu, S. Yang, P. Liang, D. Pei, J. Xu, Y. Zhang,

et al., "Prefix: Switch failure prediction in datacenter networks," *Proceedings of the ACM on Measurement and Analysis of Computing Systems*, vol. 2, no. 1, pp. 1–29, 2018. 2

[13] W. Pourmajidi, L. Zhang, A. Miranskyy, J. Steinbacher, D. Godwin, and T. Erwin, "The challenging landscape of cloud monitoring," *Knowledge Management in the Development of Data-Intensive Systems*, pp. 157–189, 2021. 3, 4

[14] H. Ren, B. Xu, Y. Wang, C. Yi, C. Huang, X. Kou, T. Xing, M. Yang, J. Tong, and Q. Zhang, "Time-series anomaly detection service at microsoft," in *Proceedings of the 25th ACM SIGKDD international conference on knowledge discovery & data mining*, pp. 3009–3017, 2019. 3

[15] Q. Yan and F. R. Yu, "Distributed denial of service attacks in software-defined networking with cloud computing," *IEEE Communications Magazine*, vol. 53, no. 4, pp. 52–59, 2015. 3

[16] K. A. Fidele and A. Hartanto, "Dos attack prevention using rule-based sniffing technique and firewall in cloud computing," in *E3S Web of Conferences*, vol. 125, p. 21004, EDP Sciences, 2019. 3

[17] R. Rajendran, S. Santhosh Kumar, Y. Palanichamy, and K. Arputharaj, "Detection of dos attacks in cloud networks using intelligent rule based classification system," *Cluster Computing*, vol. 22, pp. 423–434, 2019. 3

[18] H. Liu and A. Gegov, "Rule based systems and networks: Deterministic and fuzzy approaches," in *2016 IEEE 8th International Conference on Intelligent Systems (IS)*, pp. 316–321, IEEE, 2016. 3

[19] Z. Li, C. Luo, Y. Zhao, Y. Sun, K. Sui, X. Wang, D. Liu, X. Jin, Q. Wang, and D. Pei, "Generic and robust localization of multi-dimensional root causes," in *2019 IEEE 30th International Symposium on Software Reliability Engineering (ISSRE)*, pp. 47–57, IEEE, 2019. 3

[20] T. Barik, R. DeLine, S. Drucker, and D. Fisher, "The bones of the system: A case study of logging and telemetry at microsoft," in *2016 IEEE/ACM 38th International Conference on Software Engineering Companion (ICSE-C)*, pp. 92–101, IEEE, 2016. 3

[21] B. Chen and Z. M. Jiang, "A survey of software log instrumentation," *ACM Computing Surveys (CSUR)*, vol. 54, no. 4, pp. 1–34, 2021. 3

[22] L. Pérennou, *Virtual machine experience design: a predictive resource allocation approach for cloud infrastructures*. PhD thesis, Conservatoire national des arts et metiers-CNAM, 2019. 6

[23] M. Fowler and M. Foemmel, "Continuous integration," 2006. 7

[24] X. Zhang, Y. Xu, Q. Lin, B. Qiao, H. Zhang, Y. Dang, C. Xie, X. Yang, Q. Cheng, Z. Li, *et al.*, "Robust log-based anomaly detection on unstable log data," in *Proceedings of the 2019 27th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering*, pp. 807–817, 2019. 7, 9, 17, 19, 46, 95, 97

[25] R. Gerhards, "The Syslog Protocol." RFC 5424 (Proposed Standard), March 2009. 8, 12

[26] S. Satpathi, S. Deb, R. Srikant, and H. Yan, "Learning latent events from network message logs," *IEEE/ACM Transactions on Networking*, vol. 27, no. 4, pp. 1728–1741, 2019. 9

[27] S. Khatuya, N. Ganguly, J. Basak, M. Bharde, and B. Mitra, "Adele: Anomaly detection from event log empiricism," in *IEEE INFOCOM 2018-IEEE Conference on Computer Communications*, pp. 2114–2122, IEEE, 2018. 9

[28] V.-H. Le and H. Zhang, "Log-based anomaly detection with deep learning: How far are we?," in *Proceedings of the 44th International Conference on Software Engineering*, pp. 1356–1367, 2022. 9

[29] W. Meng, Y. Liu, Y. Zhu, S. Zhang, D. Pei, Y. Liu, Y. Chen, R. Zhang, S. Tao, P. Sun, *et al.*, "Loganomaly: Unsupervised detection of sequential and quantitative anomalies in unstructured logs.," in *IJCAI*, vol. 19, pp. 4739–4745, 2019. 9, 17, 19, 22, 46, 95, 97, 98

[30] S. Nedelkoski, J. Bogatinovski, A. Acker, J. Cardoso, and O. Kao, "Self-supervised log parsing," *arXiv preprint arXiv:2003.07905*, 2020. 9, 21, 22

[31] A. Vervaet, R. Chiky, and M. Callau-Zori, "Ustep: Unfixed search tree for efficient log parsing," in *2021 IEEE International Conference on Data Mining (ICDM)*, pp. 659–668, IEEE, 2021. 11, 37, 38, 45, 98, 100

[32] A. Vervaet, R. Chiky, and M. Callau-Zori, "Automatisation de la structuration des logs pour le cloud computing," *Extraction et Gestion des Connaissances: Actes EGC'2021*, 2021. 11, 30, 99

[33] A. Vervaet, R. Chiky, and M. Callau-Zori, "Ustep: Structuration des logs en flux grâce à un arbre de recherche évolutif," *Extraction et Gestion des Connaissances: EGC'2022*, vol. 38, 2022. 11, 38

[34] A. Vervaet, "Monilog: An automated log-based anomaly detection system for cloud computing infrastructures," in *2021 IEEE 37th International Conference on Data Engineering (ICDE)*, pp. 2739–2743, IEEE, 2021. 12, 26

[35] S. He, P. He, Z. Chen, T. Yang, Y. Su, and M. R. Lyu, "A survey on automated log analysis for reliability engineering," *ACM Computing Surveys (CSUR)*, vol. 54, no. 6, pp. 1–37, 2021. 14, 21

[36] D. A. Bhanage, A. V. Pawar, and K. Kotecha, "It infrastructure anomaly detection and failure handling: A systematic literature review focusing on datasets, log preprocessing, machine & deep learning approaches and automated tool," *IEEE Access*, vol. 9, pp. 156392–156421, 2021. 14, 17, 96

[37] A. Oliner, A. Ganapathi, and W. Xu, "Advances and challenges in log analysis," *Communications of the ACM*, vol. 55, no. 2, pp. 55–61, 2012. 14

[38] S. Di, R. Gupta, M. Snir, E. Pershey, and F. Cappello, "Logaider: A tool for mining potential correlations of hpc log events," in *2017 17th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing (CCGRID)*, pp. 442–451, IEEE, 2017. 15

[39] T. Wang, S. Byna, G. K. Lockwood, S. Snyder, P. Carns, S. Kim, and N. J. Wright, "A zoom-in analysis of i/o logs to detect root causes of i/o performance bottlenecks," in *2019 19th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing (CCGRID)*, pp. 102–111, IEEE, 2019. 15

[40] J. Tan, S. Kavulya, R. Gandhi, and P. Narasimhan, "Visual, log-based causal tracing for performance debugging of mapreduce systems," in *2010 IEEE 30th International Conference on Distributed Computing Systems*, pp. 795–806, IEEE, 2010. 15

[41] J. Dean and S. Ghemawat, "Mapreduce: simplified data processing on large clusters," *Communications of the ACM*, vol. 51, no. 1, pp. 107–113, 2008. 15

[42] J. Lu, C. Liu, F. Li, L. Li, X. Feng, and J. Xue, "Cloudraid: Detecting distributed concurrency bugs via log-mining and enhancement," *IEEE Transactions on Software Engineering*, 2020. 15

[43] M. Moh, S. Pininti, S. Doddapaneni, and T.-S. Moh, "Detecting web attacks using multi-stage log analysis," in *2016 IEEE 6th international conference on advanced computing (IACC)*, pp. 733–738, IEEE, 2016. 16

[44] M. Mimura, "Adjusting lexical features of actual proxy logs for intrusion detection," *Journal of Information Security and Applications*, vol. 50, p. 102408, 2020. 16

[45] E. Yoon and A. Squicciarini, "Toward detecting compromised mapreduce workers through log analysis," in *2014 14th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing*, pp. 41–50, IEEE, 2014. 16

[46] T. Schindler, "Anomaly detection in log data using graph databases and machine learning to defend advanced persistent threats," *arXiv preprint arXiv:1802.00259*, 2018. 16

[47] N. N. Tran, R. Sarker, and J. Hu, "An approach for host-based intrusion detection system design using convolutional neural network," in *Mobile Networks and Management: 9th International Conference, MONAMI 2017, Melbourne, Australia, December 13-15, 2017, Proceedings 9*, pp. 116–126, Springer, 2018. 16

[48] X. Gao, S. Zha, X. Li, B. Yan, X. Jing, J. Li, and J. Xu, "Incremental prediction model of disk failures based on the density metric of edge samples," *IEEE Access*, vol. 7, pp. 114285–114296, 2019. 16

[49] I. C. Chaves, M. R. P. De Paula, L. G. Leite, J. P. P. Gomes, and J. C. Machado, "Hard disk drive failure prediction method based on a bayesian network," in *2018 International Joint Conference on Neural Networks (IJCNN)*, pp. 1–7, IEEE, 2018. 16

[50] J. Shen, J. Wan, S.-J. Lim, and L. Yu, "Random-forest-based failure prediction for hard disk drives," *International Journal of Distributed Sensor Networks*, vol. 14, no. 11, p. 1550147718806480, 2018. 16

[51] A. Das, F. Mueller, and B. Rountree, "Aarohi: Making real-time node failure prediction feasible," in *2020 IEEE International Parallel and Distributed Processing Symposium (IPDPS)*, pp. 1092–1101, IEEE, 2020. 16

[52] İ. Karakurt, S. Özer, T. Ulusinan, and M. C. Ganiz, "A machine learning approach to database failure prediction," in *2017 International Conference on Computer Science and Engineering (UBMK)*, pp. 1030–1035, IEEE, 2017. 16

[53] R. Chalapathy and S. Chawla, "Deep learning for anomaly detection: A survey," *arXiv preprint arXiv:1901.03407*, 2019. 16

[54] M. E. Tschuchnig and M. Gadermayr, "Anomaly detection in medical imaging-a mini

review," in *Data Science–Analytics and Applications: Proceedings of the 4th International Data Science Conference–iDSC2021*, pp. 33–38, Springer, 2022. 16

[55] C. Baur, R. Graf, B. Wiestler, S. Albarqouni, and N. Navab, "Steganomaly: Inhibiting cyclegan steganography for unsupervised anomaly detection in brain mri," in *Medical Image Computing and Computer Assisted Intervention–MICCAI 2020: 23rd International Conference, Lima, Peru, October 4–8, 2020, Proceedings, Part II 23*, pp. 718–727, Springer, 2020. 16

[56] S. G. Popescu, D. J. Sharp, J. H. Cole, K. Kamnitsas, and B. Glocker, "Distributional gaussian process layers for outlier detection in image segmentation," in *Information Processing in Medical Imaging: 27th International Conference, IPMI 2021, Virtual Event, June 28–June 30, 2021, Proceedings 27*, pp. 415–427, Springer, 2021. 16

[57] H. E. Atlason, A. Love, S. Sigurdsson, V. Gudnason, and L. M. Ellingsen, "Unsupervised brain lesion segmentation from mri using a convolutional autoencoder," in *Medical Imaging 2019: Image Processing*, vol. 10949, pp. 372–378, SPIE, 2019. 16

[58] Z. Alaverdyan, J. Jung, R. Bouet, and C. Lartizien, "Regularized siamese neural network for unsupervised outlier detection on brain multiparametric magnetic resonance imaging: application to epilepsy lesion screening," *Medical image analysis*, vol. 60, p. 101618, 2020. 16

[59] R. Bin Sulaiman, V. Schetinin, and P. Sant, "Review of machine learning approach on credit card fraud detection," *Human-Centric Intelligent Systems*, vol. 2, no. 1-2, pp. 55–68, 2022. 16

[60] S. Shirgave, C. Awati, R. More, and S. Patil, "A review on credit card fraud detection using machine learning," *International Journal of Scientific & technology research*, vol. 8, no. 10, pp. 1217–1220, 2019. 16

[61] Z. Ahmad, A. Shahid Khan, C. Wai Shiang, J. Abdullah, and F. Ahmad, "Network

intrusion detection system: A systematic study of machine learning and deep learning approaches," *Transactions on Emerging Telecommunications Technologies*, vol. 32, no. 1, p. e4150, 2021. 16

[62] N. Chaabouni, M. Mosbah, A. Zemmari, C. Sauvignac, and P. Faruki, "Network intrusion detection for iot security based on learning techniques," *IEEE Communications Surveys & Tutorials*, vol. 21, no. 3, pp. 2671–2701, 2019. 16

[63] S. He, J. Zhu, P. He, and M. R. Lyu, "Experience report: System log analysis for anomaly detection," in *2016 IEEE 27th International Symposium on Software Reliability Engineering (ISSRE)*, pp. 207–218, IEEE, 2016. 17, 32

[64] Y. Liang, Y. Zhang, H. Xiong, and R. Sahoo, "Failure prediction in ibm bluegene/l event logs," in *Seventh IEEE International Conference on Data Mining (ICDM 2007)*, pp. 583–588, IEEE, 2007. 17, 18, 97

[65] W. Xu, L. Huang, A. Fox, D. Patterson, and M. Jordan, "Largescale system problem detection by mining console logs," *Proceedings of SOSP'09*, 2009. 17, 18, 97

[66] A. Farzad and T. A. Gulliver, "Unsupervised log message anomaly detection," *ICT Express*, vol. 6, no. 3, pp. 229–237, 2020. 17, 20, 97

[67] J.-G. Lou, Q. Fu, S. Yang, Y. Xu, and J. Li, "Mining invariants from console logs for system problem detection.," in *USENIX Annual Technical Conference*, pp. 1–14, 2010. 17, 18, 97

[68] X. Li, P. Chen, L. Jing, Z. He, and G. Yu, "Swisslog: Robust and unified deep learning based log anomaly detection for diverse faults," in *2020 IEEE 31st International Symposium on Software Reliability Engineering (ISSRE)*, pp. 92–103, IEEE, 2020. 17, 19, 97

[69] Q. Lin, H. Zhang, J.-G. Lou, Y. Zhang, and X. Chen, "Log clustering based problem iden-

tification for online service systems," in *2016 IEEE/ACM 38th International Conference on Software Engineering Companion (ICSE-C)*, pp. 102–111, IEEE, 2016. 17, 18, 25, 97

[70] H. Ott, J. Bogatinovski, A. Acker, S. Nedelkoski, and O. Kao, "Robust and transferable anomaly detection in log data using pre-trained language models," in *2021 IEEE/ACM International Workshop on Cloud Intelligence (CloudIntelligence)*, pp. 19–24, IEEE, 2021. 17, 97

[71] M. Du, F. Li, G. Zheng, and V. Srikumar, "Deeplog: Anomaly detection and diagnosis from system logs through deep learning," in *Proceedings of the 2017 ACM SIGSAC conference on computer and communications security*, pp. 1285–1298, 2017. 17, 19, 25, 28, 95, 97

[72] L. Yang, J. Chen, Z. Wang, W. Wang, J. Jiang, X. Dong, and W. Zhang, "Semi-supervised log-based anomaly detection via probabilistic label estimation," in *2021 IEEE/ACM 43rd International Conference on Software Engineering (ICSE)*, pp. 1448–1460, IEEE, 2021. 17, 19, 97

[73] M. Wurzenberger, F. Skopik, M. Landauer, P. Greitbauer, R. Fiedler, and W. Kastner, "Incremental clustering for semi-supervised anomaly detection applied on log data," in *Proceedings of the 12th International Conference on Availability, Reliability and Security*, pp. 1–6, 2017. 17, 18, 97

[74] H. Guo, S. Yuan, and X. Wu, "Logbert: Log anomaly detection via bert," in *2021 international joint conference on neural networks (IJCNN)*, pp. 1–8, IEEE, 2021. 17, 20, 97

[75] S. Lu, X. Wei, Y. Li, and L. Wang, "Detecting anomaly in big data system logs using convolutional neural network," in *2018 IEEE 16th Intl Conf on Dependable, Autonomic and Secure Computing, 16th Intl Conf on Pervasive Intelligence and Computing, 4th Intl Conf on Big Data Intelligence and Computing and Cyber Science and Technology Congress (DASC/PiCom/DataCom/CyberSciTech)*, pp. 151–158, IEEE, 2018. 17, 19, 97

[76] B. Xia, Y. Bai, J. Yin, Y. Li, and J. Xu, "Loggan: a log-level generative adversarial network for anomaly detection using permutation event modeling," *Information Systems Frontiers*, vol. 23, pp. 285–298, 2021. 17, 20, 97

[77] S. He, Q. Lin, J.-G. Lou, H. Zhang, M. R. Lyu, and D. Zhang, "Identifying impactful service system problems via log analysis," in *Proceedings of the 2018 26th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering*, pp. 60–70, 2018. 17, 97

[78] X. Duan, S. Ying, W. Yuan, H. Cheng, and X. Yin, "Qllog: A log anomaly detection method based on q-learning algorithm," *Information Processing & Management*, vol. 58, no. 3, p. 102540, 2021. 17, 97

[79] S. Nedelkoski, J. Bogatinovski, A. Acker, J. Cardoso, and O. Kao, "Self-attentive classification-based anomaly detection in unstructured logs," in *2020 IEEE International Conference on Data Mining (ICDM)*, pp. 1196–1201, IEEE, 2020. 17, 20, 97

[80] V.-H. Le and H. Zhang, "Log-based anomaly detection without log parsing," in *2021 36th IEEE/ACM International Conference on Automated Software Engineering (ASE)*, pp. 492–504, IEEE, 2021. 17, 97

[81] Z. Wang, J. Tian, H. Fang, L. Chen, and J. Qin, "Lightlog: A lightweight temporal convolutional network for log anomaly detection on the edge," *Computer Networks*, vol. 203, p. 108616, 2022. 17, 97

[82] Z. Chen, J. Liu, W. Gu, Y. Su, and M. R. Lyu, "Experience report: Deep learning-based system log analysis for anomaly detection," *arXiv preprint arXiv:2107.05908*, 2021. 17

[83] A. Ramachandra and S. Bhattacharya, "Literature survey on log-based anomaly detection framework in cloud," *Computational Intelligence in Pattern Recognition: Proceedings of CIPR 2020*, pp. 143–153, 2020. 17

[84] S. Zhang, W. Meng, J. Bu, S. Yang, Y. Liu, D. Pei, J. Xu, Y. Chen, H. Dong, X. 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)*, pp. 1–10, IEEE, 2017. 19

[85] J. Devlin, M.-W. Chang, K. Lee, and K. Toutanova, "Bert: Pre-training of deep bidirectional transformers for language understanding," *arXiv preprint arXiv:1810.04805*, 2018. 19, 20, 22

[86] Z. Li, F. Liu, W. Yang, S. Peng, and J. Zhou, "A survey of convolutional neural networks: analysis, applications, and prospects," *IEEE transactions on neural networks and learning systems*, 2021. 19

[87] R. Ren, J. Cheng, Y. Yin, J. Zhan, L. Wang, J. Li, and C. Luo, "Deep convolutional neural networks for log event classification on distributed cluster systems," in *2018 IEEE International Conference on Big Data (Big Data)*, pp. 1639–1646, IEEE, 2018. 19

[88] F. T. Liu, K. M. Ting, and Z.-H. Zhou, "Isolation forest," in *2008 eighth ieee international conference on data mining*, pp. 413–422, IEEE, 2008. 20, 67

[89] S. Kabinna, C.-P. Bezemer, W. Shang, M. D. Syer, and A. E. Hassan, "Examining the stability of logging statements," *Empirical Software Engineering*, vol. 23, no. 1, pp. 290–333, 2018. 20

[90] T. Zhang, H. Qiu, G. Castellano, M. Rifai, C. S. Chen, and F. Pianese, "System log parsing: A survey," *IEEE Transactions on Knowledge and Data Engineering*, 2023. 21

[91] Q. Fu, J.-G. Lou, Y. Wang, and J. Li, "Execution anomaly detection in distributed systems through unstructured log analysis," in *2009 ninth IEEE international conference on data mining*, pp. 149–158, IEEE, 2009. 21

[92] L. Tang, T. Li, and C.-S. Perng, "Logsig: Generating system events from raw textual logs,"

in *Proceedings of the 20th ACM international conference on Information and knowledge management*, pp. 785–794, 2011. 21

[93] H. Hamooni, B. Debnath, J. Xu, H. Zhang, G. Jiang, and A. Mueen, "Logmine: Fast pattern recognition for log analytics," in *Proceedings of the 25th ACM International on Conference on Information and Knowledge Management*, pp. 1573–1582, 2016. 21

[94] A. A. Makanju, A. N. Zincir-Heywood, and E. E. Milios, "Clustering event logs using iterative partitioning," in *Proceedings of the 15th ACM SIGKDD international conference on Knowledge discovery and data mining*, pp. 1255–1264, 2009. 21, 23

[95] R. Vaarandi, "A data clustering algorithm for mining patterns from event logs," in *Proceedings of the 3rd IEEE Workshop on IP Operations & Management (IPOM 2003)(IEEE Cat. No. 03EX764)*, pp. 119–126, IEEE, 2003. 21

[96] M. Nagappan and M. A. Vouk, "Abstracting log lines to log event types for mining software system logs," in *2010 7th IEEE Working Conference on Mining Software Repositories (MSR 2010)*, pp. 114–117, IEEE, 2010. 21

[97] R. Vaarandi and M. Pihelgas, "Logcluster-a data clustering and pattern mining algorithm for event logs," in *2015 11th International conference on network and service management (CNSM)*, pp. 1–7, IEEE, 2015. 21

[98] Z. M. Jiang, A. E. Hassan, P. Flora, and G. Hamann, "Abstracting execution logs to execution events for enterprise applications (short paper)," in *2008 The Eighth International Conference on Quality Software*, pp. 181–186, IEEE, 2008. 21

[99] H. Dai, H. Li, W. Shang, T.-H. Chen, and C.-S. Chen, "Logram: Efficient log parsing using n-gram dictionaries," *arXiv preprint arXiv:2001.03038*, 2020. 21, 23

[100] M. Mizutani, "Incremental mining of system log format," in *2013 IEEE International Conference on Services Computing*, pp. 595–602, IEEE, 2013. 22, 46, 98

[101] K. Shima, "Length matters: Clustering system log messages using length of words," *arXiv preprint arXiv:1611.03213*, 2016. 22, 46, 98

[102] M. Du and F. Li, "Spell: Streaming parsing of system event logs," in *2016 IEEE 16th International Conference on Data Mining (ICDM)*, pp. 859–864, IEEE, 2016. 22, 23, 47, 98

[103] P. He, J. Zhu, Z. Zheng, and M. R. Lyu, "Drain: An online log parsing approach with fixed depth tree," in *2017 IEEE International Conference on Web Services (ICWS)*, pp. 33–40, IEEE, 2017. 22, 28, 46, 98

[104] P. He, J. Zhu, P. Xu, Z. Zheng, and M. R. Lyu, "A directed acyclic graph approach to online log parsing," 2018. 22, 98

[105] H. Liang, L. Song, J. Wang, L. Guo, X. Li, and J. Liang, "Robust unsupervised anomaly detection via multi-time scale dcgans with forgetting mechanism for industrial multivariate time series," *Neurocomputing*, vol. 423, pp. 444–462, 2021. 22, 98

[106] P. He, J. Zhu, S. He, J. Li, and M. R. Lyu, "Towards automated log parsing for large-scale log data analysis," *IEEE Transactions on Dependable and Secure Computing*, vol. 15, no. 6, pp. 931–944, 2017. 23

[107] A. Agrawal, R. Karlupia, and R. Gupta, "Logan: A distributed online log parser," in *2019 IEEE 35th International Conference on Data Engineering (ICDE)*, pp. 1946–1951, IEEE, 2019. 23, 53

[108] P. He, J. Zhu, S. He, J. Li, and M. R. Lyu, "An evaluation study on log parsing and its use in log mining," in *2016 46th annual IEEE/IFIP international conference on dependable systems and networks (DSN)*, pp. 654–661, IEEE, 2016. 23, 30

[109] S. Messaoudi, A. Panichella, D. Bianculli, L. Briand, and R. Sasnauskas, "A search-based approach for accurate identification of log message formats," in *Proceedings of the 26th Conference on Program Comprehension*, pp. 167–177, 2018. 23

[110] S. He, J. Zhu, P. He, and M. R. Lyu, "Loghub: a large collection of system log datasets towards automated log analytics," *arXiv preprint arXiv:2008.06448*, 2020. 25, 28, 32, 45, 46

[111] A. Oliner and J. Stearley, "What supercomputers say: A study of five system logs," in *37th annual IEEE/IFIP international conference on dependable systems and networks (DSN'07)*, pp. 575–584, IEEE, 2007. 25

[112] W. Xu, L. Huang, A. Fox, D. Patterson, and M. I. Jordan, "Detecting large-scale system problems by mining console logs," in *Proceedings of the ACM SIGOPS 22nd symposium on Operating systems principles*, pp. 117–132, 2009. 25

[113] Y. Fu, M. Yan, Z. Xu, X. Xia, X. Zhang, and D. Yang, "An empirical study of the impact of log parsers on the performance of log-based anomaly detection," *Empirical Software Engineering*, vol. 28, no. 1, pp. 1–39, 2023. 29

[114] F. Archetti and A. Candelieri, *Bayesian optimization and data science*, vol. 849. Springer, 2019. 33

[115] E. J. Ghomi, A. M. Rahmani, and N. N. Qader, "Load-balancing algorithms in cloud computing: A survey," *Journal of Network and Computer Applications*, vol. 88, pp. 50–71, 2017. 55

[116] S. K. Mishra, B. Sahoo, and P. P. Parida, "Load balancing in cloud computing: A big picture," *Journal of King Saud University-Computer and Information Sciences*, vol. 32, no. 2, pp. 149–158, 2020. 55

[117] B. Debnath, M. Solaimani, M. A. G. Gulzar, N. Arora, C. Lumezanu, J. Xu, B. Zong, H. Zhang, G. Jiang, and L. Khan, "Loglens: A real-time log analysis system," in *2018 IEEE 38th international conference on distributed computing systems (ICDCS)*, pp. 1052–1062, IEEE, 2018. 66

[118] D.-Q. Zou, H. Qin, and H. Jin, "Uilog: Improving log-based fault diagnosis by log analysis," *Journal of computer science and technology*, vol. 31, no. 5, pp. 1038–1052, 2016. 67

[119] A. Pande and V. Ahuja, "Weac: Word embeddings for anomaly classification from event logs," in *2017 IEEE International Conference on Big Data (Big Data)*, pp. 1095–1100, IEEE, 2017. 67

[120] R. J. Campello, D. Moulavi, A. Zimek, and J. Sander, "Hierarchical density estimates for data clustering, visualization, and outlier detection," *ACM Transactions on Knowledge Discovery from Data (TKDD)*, vol. 10, no. 1, pp. 1–51, 2015. 67

[121] W. Meng, Y. Liu, S. Zhang, D. Pei, H. Dong, L. Song, and X. Luo, "Device-agnostic log anomaly classification with partial labels," in *2018 IEEE/ACM 26th International Symposium on Quality of Service (IWQoS)*, pp. 1–6, IEEE, 2018. 67

[122] S. Hochreiter and J. Schmidhuber, "Long short-term memory," *Neural computation*, vol. 9, no. 8, pp. 1735–1780, 1997. 75

[123] H. Nguyen, K. P. Tran, S. Thomassey, and M. Hamad, "Forecasting and anomaly detection approaches using lstm and lstm autoencoder techniques with the applications in supply chain management," *International Journal of Information Management*, vol. 57, p. 102282, 2021. 75

[124] P. Malhotra, A. Ramakrishnan, G. Anand, L. Vig, P. Agarwal, and G. Shroff, "Lstm-based encoder-decoder for multi-sensor anomaly detection," *arXiv preprint arXiv:1607.00148*, 2016. 75

[125] K. Hundman, V. Constantinou, C. Laporte, I. Colwell, and T. Soderstrom, "Detecting spacecraft anomalies using lstms and nonparametric dynamic thresholding," in *Proceedings of the 24th ACM SIGKDD international conference on knowledge discovery & data mining*, pp. 387–395, 2018. 75

[126] C. Tofallis, "A better measure of relative prediction accuracy for model selection and model estimation," *Journal of the Operational Research Society*, vol. 66, no. 8, pp. 1352–1362, 2015. 77, 79

[127] B. E. Flores, "A pragmatic view of accuracy measurement in forecasting," *Omega*, vol. 14, no. 2, pp. 93–98, 1986. 84

[128] J. Svacina, J. Raffety, C. Woodahl, B. Stone, T. Cerny, M. Bures, D. Shin, K. Frajtak, and P. Tisnovsky, "On vulnerability and security log analysis: A systematic literature review on recent trends," in *Proceedings of the International Conference on Research in Adaptive and Convergent Systems*, pp. 175–180, 2020. 91

[129] R. Zuech, T. M. Khoshgoftaar, and R. Wald, "Intrusion detection and big heterogeneous data: a survey," *Journal of Big Data*, vol. 2, no. 1, pp. 1–41, 2015. 91

[130] S. Nedelkoski, J. Bogatinovski, A. Acker, J. Cardoso, and O. Kao, "Self-attentive classification-based anomaly detection in unstructured logs," in *2020 IEEE International Conference on Data Mining (ICDM)*, pp. 1196–1201, 2020.

[131] V. Anitha and P. Isakki, "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, 2016.

[132] M. Awad and D. A. Menascé, "Automatic workload characterization using system log analysis," in *Computer Measurement Group Conference on Performance and Capacity, San Antonio, TX*, 2015.

[133] T. Kimura, A. Watanabe, T. Toyono, and K. Ishibashi, "Proactive failure detection learning generation patterns of large-scale network logs," *IEICE Transactions on Communications*, 2018.

[134] S. Lu, B. Rao, X. Wei, B. Tak, L. Wang, and L. Wang, "Log-based abnormal task detection

and root cause analysis for spark," in *2017 IEEE International Conference on Web Services (ICWS)*, pp. 389–396, IEEE, 2017.

[135] J. Zhu, P. He, Q. Fu, H. Zhang, M. R. Lyu, and D. Zhang, "Learning to log: Helping developers make informed logging decisions," in *2015 IEEE/ACM 37th IEEE International Conference on Software Engineering*, vol. 1, pp. 415–425, IEEE, 2015.

[136] Q. Fu, J.-G. Lou, Q. Lin, R. Ding, D. Zhang, and T. Xie, "Contextual analysis of program logs for understanding system behaviors," in *2013 10th Working Conference on Mining Software Repositories (MSR)*, pp. 397–400, IEEE, 2013.

[137] S. Kobayashi, K. Fukuda, and H. Esaki, "Mining causes of network events in log data with causal inference," in *2017 IFIP/IEEE Symposium on Integrated Network and Service Management (IM)*, pp. 45–53, IEEE, 2017.

[138] S. Jha, S. Cui, S. Banerjee, T. Xu, J. Enos, M. Showerman, Z. T. Kalbarczyk, and R. K. Iyer, "Live forensics for hpc systems: a case study on distributed storage systems," in *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*, pp. 1–16, 2020.

[139] D. Ohana, B. Wassermann, N. Dupuis, E. Kolodner, E. Raichstein, and M. Malka, "Hybrid anomaly detection and prioritization for network logs at cloud scale," in *Proceedings of the Seventeenth European Conference on Computer Systems*, pp. 236–250, 2022.

[140] Y. Liu, X. Zhang, S. He, H. Zhang, L. Li, Y. Kang, Y. Xu, M. Ma, Q. Lin, Y. Dang, *et al.*, "Uniparser: A unified log parser for heterogeneous log data," in *Proceedings of the ACM Web Conference 2022*, pp. 1893–1901, 2022.

[141] D. El-Masri, F. Petrillo, Y.-G. Guéhéneuc, A. Hamou-Lhadj, and A. Bouziane, "A systematic literature review on automated log abstraction techniques," *Information and Software Technology*, vol. 122, p. 106276, 2020.