LogAssist: Assisting Log Analysis Through Log Summarization

Steven Locke

A Thesis

in

The Department

of

Computer Science and Software Engineering

Presented in Partial Fulfillment of the Requirements

for the Degree of

Master of Applied Science (Software Engineering) at

Concordia University

Montréal, Québec, Canada

August 2021

CONCORDIA UNIVERSITY

School of Graduate Studies

This is to certify that the thesis prepared

By: **Steven Locke**

Entitled: **LogAssist: Assisting Log Analysis Through Log Summarization**

and submitted in partial fulfillment of the requirements for the degree of

Master of Applied Science (Software Engineering)

complies with the regulations of this University and meets the accepted standards with respect to originality and quality.

Signed by the Final Examining Committee:

_____ Chair
Dr. Juergen Rilling

_____ Examiner
Dr. Juergen Rilling

_____ Examiner
Dr. Nikolaos Tsantalis

_____ Supervisor
Dr. Tse-Hsun Chen

Approved by _____
Dr. Leila Kosseim, Graduate Program Director
Department of Computer Science and Software Engineering

August 9, 2021 _____
Dr. Mourad Debbabi, Dean
Faculty of Engineering and Computer Science

Abstract

LogAssist: Assisting Log Analysis Through Log Summarization

Steven Locke

Logs contain valuable information about the runtime behaviors of software systems. Thus, practitioners rely on logs for various tasks such as debugging, system comprehension, and anomaly detection. However, logs are difficult to analyze due to their unstructured nature and large size. In this thesis, we propose a novel approach called *LogAssist* that assists practitioners with log analysis. *LogAssist* provides an organized and concise view of logs by first grouping logs into event sequences (i.e., workflows), which better illustrate the system runtime execution paths. Then, *LogAssist* compresses the log events in workflows by hiding consecutive events and applying n-gram modeling to identify common event sequences. We evaluated *LogAssist* on logs generated by one enterprise and two open source systems. We find that *LogAssist* can reduce the number of log events that practitioners need to investigate by up to 99%. Through a user study with 19 participants, we find that *LogAssist* can assist practitioners by reducing the time required for log analysis tasks by an average of 40%. The participants also rated *LogAssist* an average of 4.53 out of 5 for improving their experiences of performing log analysis. Finally, we document our experiences and lessons learned from developing and adopting *LogAssist* in practice. We believe that *LogAssist* and our reported experiences may lay the basis for future analysis and interactive exploration on logs.

Acknowledgments

First, and foremost, I would like to take this opportunity to express my sincere gratitude towards my supervisor Dr. Tse-Hsun (Peter) Chen for his guidance, encouragement, and contributions during my research journey. I feel fortunate to have had him as my supervisor and appreciate everything he has taught me. I would also like to extend my gratitude to Dr. Weiyi Shang, Dr. Heng Li, Dr. Jinqiu Yang, Dr. Nikos Tsantalis and Dr. Bram Adams for their insight, guidance, and collaboration throughout my master's degree.

I would also like to extend my thanks to my undergraduate professors Dr. Leila Kosseim, Dr. Constantinos Constantinides and Dr. Aiman Hanna. Each of these individuals have had a profound impact on me and provided me with guidance, mentorship, and support, while challenging me to be the best that I can be. If not for their passion and enthusiasm, I might not have even chosen to pursue graduate studies.

From the very beginning, my fellow lab members from the SPEAR lab, and neighbouring SENSE lab have been there to support me and set high standards for effort and quality. I am very happy to have had them share this journey with me and make lasting memories with them.

I would like to dedicate my work to my parents and thank them for their continuous support throughout my life. Without them, this thesis would not have been possible.

Related Publications

This thesis is related to the following publication:

- Steven Locke, Heng Li, Tse-Hsun (Peter) Chen, Weiyi Shang and Wei Liu. LogAssist: Assisting Log Analysis Through Log Summarization. This work was accepted for publication in IEEE Transactions on Software Engineering 2021.

My contribution: Drafting the research plan, conceiving approach, collecting and analyzing the data, implementing tool, designing user study, collecting and analyzing results, writing and polishing the paper drafts.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

Software systems generate logs during field operations or in-house testing. Such logs contain rich information about the runtime behaviors of software systems (Barik, DeLine, Drucker, & Fisher, 2016; Fu et al., 2014; Li, Shang, Adams, Sayagh, & Hassan, 2020). Therefore, logs are widely leveraged by practitioners in software development, operation, and maintenance tasks, such as failure diagnosis (*Automated Root Cause Analysis for Spark Application Failures - O'Reilly Media*, 2017; Fu et al., 2013; Yuan et al., 2010), anomaly detection (Fu, Lou, Wang, & Li, 2009; S. He et al., 2018; Jiang, Hassan, Hamann, & Flora, 2008b; Lou, Fu, Yang, Xu, & Li, 2010; Xu, Huang, Fox, Patterson, & Jordan, 2009a, 2009b), performance analysis (Chow, Meisner, Flinn, Peek, & Wenisch, 2014; Ding et al., 2015; Nagaraj, Killian, & Neville, 2012; Yao, de Pádua, et al., 2020), and system comprehension (Fu et al., 2013; Shang et al., 2013).

Despite their importance, the enormous sizes (e.g., tens or hundreds of gigabytes) of logs (A. J. Oliner & Stearley, 2007; Schroeder & Gibson, 2007) have become a major obstacle for logs analysis (Barik et al., 2016; Cito, Leitner, Fritz, & Gall, 2015; Li et al., 2020; A. Oliner, Ganapathi, & Xu, 2012; Yuan et al., 2010). In particular, analyzing large-scale log data usually faces the following challenges:

- **Unstructured logs.** Logs are unstructured data that consist of some natural language text and a few dynamic values (P. He, Chen, He, & Lyu, 2018; Yao, Li, Shang, & Hassan, 2020). Thus, it is challenging to automatically parse and analyze logs.

- **Intermixed event sequences.** Different event sequences (e.g., the sequence of events associated with a user login) are intermixed with each other, making it difficult for practitioners to understand the system runtime behaviors or identify the event sequences that may lead to a runtime issue (Yuan et al., 2010).

- **Rapidly growing log size.** Large-scale systems (e.g., cloud platforms) generate tens of gigabytes to terabytes of logs daily (Cito et al., 2015; Li et al., 2020; Reiss, Wilkes, & Hellerstein, 2011), making it challenging to manage and analyze such large-scale logs.

Prior work proposes approaches to address these challenges to a certain extent. To address the challenge related to the unstructured nature of logs, prior work proposes approaches for automatically parsing raw logs into structured forms (P. He, Zhu, Zheng, & Lyu, 2017; Jiang, Hassan, Hamann, & Flora, 2008a). However, prior work rarely explores the challenges related to intermixed event sequences. To address the challenge related to the large size of logs, prior work proposes approaches for compressing logs (Liu et al., 2019; Yao, Li, et al., 2020). However, such log compression approaches only aim to save storage space while not being able to provide assistance when logs are analyzed in practice. Commercial log analytic platforms like Splunk (Splunk, 2017) and ELK (Elastic, n.d.) also allow practitioners to efficiently manage and analyze large-scale logs (e.g., search for keywords) by leveraging distributed storage. However, such log analytic platforms are unable to provide detailed insights into the specific event sequences associated with such keywords.

In this work, we propose *LogAssist*, a novel approach for assisting practitioners with log analysis, which aims to address all the three above-mentioned challenges. First, *LogAssist* parses the raw logs into abstracted log events (i.e., addressing the challenge related to unstructured logs). Then, *LogAssist* untangles the raw logs into meaningful event sequences (i.e., workflows) using certain grouping IDs commonly available in logs, to address the challenge related to intermixed event sequences. Finally, *LogAssist* leverages n-gram models to identify common event sequences, and further uses the identified sequences to compress the logs into a much more concise representation (i.e., addressing the challenge related to the large size of logs). In addition, *LogAssist* allows practitioners to expand and explore the compressed form on demand, providing practitioners the flexibility to access the complete information in the logs. We evaluate *LogAssist* on logs from one

enterprise and two open source systems. We study the effectiveness of *LogAssist* both quantitatively and qualitatively, by answering three research questions (RQs):

RQ 1: *How well can logs be compressed into re-occurring event sequences?* We quantitatively examine how effectively *LogAssist* can compress raw logs into concise representations.

RQ 2: *How much can* LogAssist *reduce the volume of logs needed to be examined in log analysis tasks?* We quantitatively examine how effectively *LogAssist* can reduce the number of log lines that need to be examined by practitioners when performing log analysis tasks.

RQ 3: *How much can* LogAssist *help improve users' log analysis experiences?* We conduct a user study to understand how well *LogAssist* can improve users' experiences when performing log analysis tasks over using raw logs alone.

Our results show that *LogAssist* can compress the raw logs into a much more concise representation, while allowing practitioners to access the complete information of logs only when necessary. *LogAssist* significantly simplifies log analysis tasks and improves practitioners' log analysis experiences. We document our experiences and lessons learned from developing and adopting our approach in practice, which can provide insights for researchers and practitioners who wish to develop similar tools to assist with log analysis tasks. *LogAssist* can be leveraged as a basis and starting point to further advance interactive log analysis techniques.

1.1 Research Statement

Prior research studies techniques to process, compress, and store logs, while existing tools aim to help efficiently manage and analyze logs. However, prior work and tools rarely explore or address the challenges related to intermixed event sequences contained within logs. In this thesis, we study the effectiveness of applying natural language processing techniques to logs, to assist practitioners by providing detailed insights into the specific event sequences contained within logs.

Natural language processing techniques, such as n-gram modeling, can be used to effectively summarize logs by extracting reoccuring sequences, reduce the volume of logs needed to be examined, and improve users' experiences during log analysis.

1.2 Thesis Contributions

In this thesis, we propose a novel approach called *LogAssist*, which transforms logs, and presents them to practitioners in a more organized and practical view, for the purpose of facilitating log analysis tasks. The novel contributions of *LogAssist* are found primarily in the workflow creation, workflow reduction, and log reconstruction steps of our approach.

Like *LogAssist*, many existing approaches leverage log abstraction as an initial step to parse and process logs into a practical form which can be used in further steps. For the log abstraction step, we apply an existing state-of-the-art log abstraction approach to parse, abstract, and categorize log lines.

In the workflow creation step, we start by following existing work to group log lines together into workflows using grouping IDs often provided in logs. Then, we expand on prior work by proposing to separate workflows further, to account for the possible re-using of such grouping IDs. We propose the use of a popular signal processing algorithm to achieve this separation.

In the workflow reduction step, we start by following existing work to reduce workflows through the collapsing of consecutive duplicate events. Then, we expand on prior work by applying statistical techniques, namely n-gram modeling, in conjunction with existing techniques of collapsing consecutive duplicate events. In this way, *LogAssist* is able to identify and reduce re-occurring sequences of events, including those that contain multiple different event types, which existing approaches are unable to do. Furthermore, by applying these techniques iteratively in conjunction, *LogAssist* is capable of reducing entire repeating event sequences, achieving a much more concise reduced representation than the current state-of-the-art. By categorizing workflows based on their shared reduced representation, *LogAssist* is able to achieve significantly higher levels for grouping of common workflows, by identifying variances in unique workflow types that existing state-of-the-art approaches are unable to detect.

While prior works share common steps of log abstraction, workflow creation, and workflow reduction, these approaches aim to solve very different challenges than *LogAssist*. Prior studies often focus on anomaly detection, or identifying deployment problems through the comparison of workflow types between testing and production environments. Due to the differences in goals, such

approaches do not include a log reconstruction step to rebuild logs. As the goal of *LogAssist* is to transform logs into a representation that can facilitate log analysis tasks, the final step of our approach is log reconstruction. In this step, logs are reconstructed into an organized, flexible, and dynamic representation, with additional insights and statistics provided for the workflows. Commercial log analytics platforms can provide insights into individual events, or keywords, but are unable to provide details regarding entire sequences of events. With *LogAssist*, the reconstructed logs provide practitioners with insights into such sequences of events.

We propose *LogAssist* as a starting point to further advance interactive log analysis techniques and tools, to assist practitioners with log analysis. While static logs of intermixed events limit the usefulness and application of logs, such interactive log analysis techniques and tools can transform logs into flexible forms that can be tailored to suit various log analysis tasks at the discretion of the user, and provide additional information to assist with tasks.

1.3 Organization of the Thesis

Chapter 2 provides motivating examples. Chapter 3 describes the design and implementation of our approach. Chapter 4 presents the evaluation results. Chapter 5 discusses the lessons that we learned from developing and adopting our approach. Chapter 6 outlines the possible threats to the validity of our findings. Chapter 7 discusses related work. Chapter 8 concludes this thesis.

Chapter 2

Motivating Examples

To illustrate the challenges that practitioners face during log analysis, we present motivating examples of using logs in three hypothetical, yet realistic situations on a large-scale enterprise system. The system is composed of several large components. Each component can be distributed in different environments and serve different purposes.

2.1 Situation one: Anomaly detection after load testing.

Dave is a load testing specialist. Dave's main day-to-day job is to test the behavior of the system under load before the system is released to the customers. Dave designs a 48-hour test that simulates real-world user usages. After running the test, Dave needs to confirm whether there exist any anomalous behaviors that occurred during the test. Such a task is typically done by analyzing the logs that are generated during the test. However, due to the scale of the system and the lengthy nature of the test, the generated logs are of tremendous size. As it is impossible for Dave to manually analyze gigabytes or even terabytes of logs, Dave uses simple keyword search (e.g., *error* or *exception*) to find problematic log lines (T.-H. Chen et al., 2017; Jiang & Hassan, 2015; Shang et al., 2013). Unfortunately, the search results still return thousands of problematic log lines. Dave needs to manually investigate not only these log lines but also the related log events to uncover the system execution that led to the problem (A. Chen, Chen, & Wang, 2021; A. R. Chen, Chen, & Wang, 2021; LaToza & Myers, 2010; Yuan et al., 2010). As the resulting logs contain intermixed information

from both normal and abnormal system behaviour, Dave encounters challenges when analyzing an enormous amount of unstructured logs. It is challenging and difficult for Dave to manually identify which events correspond to specific execution sequences to understand the system behaviour and diagnose possible anomalous event sequences.

2.2 Situation two: Recovering common user behaviors.

From time to time, Dave also needs to update the design of the load test to reflect changes in user behaviors and system functionality. Hence, Dave needs to recover the common user behaviors by analyzing the logs generated by end users in the deployed system. Such recovered common user behaviors can later be integrated into the design of the updated load tests. Similarly, Dave relies on using keywords (e.g., *log in* or *checkout*) that are related to the key functionality to search for common user behaviors. However, due to the complexity of the system, such keyword searches may return inaccurate estimation on the executed loads. For example, one user action may result in multiple log lines containing the same keyword, or some keywords may be removed from the logs as the system evolves. Dave faces the challenge of manually summarizing the logs and identifying the corresponding user actions. These logs are large in scale, and may be interwoven and contain many repetitions, which makes the analysis even more difficult.

2.3 Situation three: Identifying the root causes of system runtime issues.

Alice is a senior developer in the team. Alice's main duty is to develop new features and maintain the quality of the code. When a system runtime issue occurs, Alice needs to investigate the issue and find the root cause in the code. In particular, Alice needs to examine the logs that may provide clues for the system runtime activities (i.e., event sequences that represent the system execution path) that led to the runtime issue. However, leveraging the raw logs to identify such clues is challenging (A. R. Chen et al., 2021; Yuan et al., 2010). As many execution workflows intermix with others in the logs, it is difficult to manually examine the logs and find the corresponding events

that lead to a runtime issue.

2.4 Challenges observed during the above-mentioned situations.

Logs in their nature are unstructured and disorganized. Although often written in the form of human-readable text, manually exploring logs in practice is counter-productive and often impossible due to the massive size of logs. Therefore, for the practitioners who depend on logs on a daily basis, there is an urgent need for automated techniques that can summarize logs for further manual exploration, while preserving the valuable information contained within the logs. In order to assist our industrial partner in addressing such challenges, we design an approach that can automatically summarize a large number of logs and assist practitioners with various log analysis tasks.

Chapter 3

The Design of *LogAssist*

In this chapter, we describe our approach, *LogAssist*, which transforms raw logs into a concise form that is more convenient for practitioners to browse and analyze.

Figure 3.1 illustrates the overall process of our approach with a running example. First, *LogAssist* parses the raw logs into structured logs (i.e., log events). Then, the log events are grouped by grouping IDs (e.g., thread IDs) to form workflows. Next, *LogAssist* compresses the log events in each workflow into a more concise representation using n-grams. Finally, *LogAssist* can reconstruct the original logs from the compressed form. We implement *LogAssist* as a prototype which helps practitioners with log analysis. We explain the detailed steps of *LogAssist* below.

3.1 Log Abstraction

Raw logs are unstructured text that contain both static and dynamic information. Such unstructured logs first need to be converted into a structured form to perform subsequent analysis (T.-H. Chen et al., 2017; Xu et al., 2009a; Zhu et al., 2019). Log abstraction is widely used to categorize raw log lines (T.-H. Chen et al., 2017; Du, Li, Zheng, & Srikumar, 2017; Shang et al., 2013; Syer et al., 2013, 2014) which involves parsing log files by separating the static and dynamic components of each log line, and assigning a common event ID to lines which share a common template for the remaining static components. This process allows for categorizing log lines by representing a line by the resulting event ID of the log abstraction tool results. By categorizing and representing log

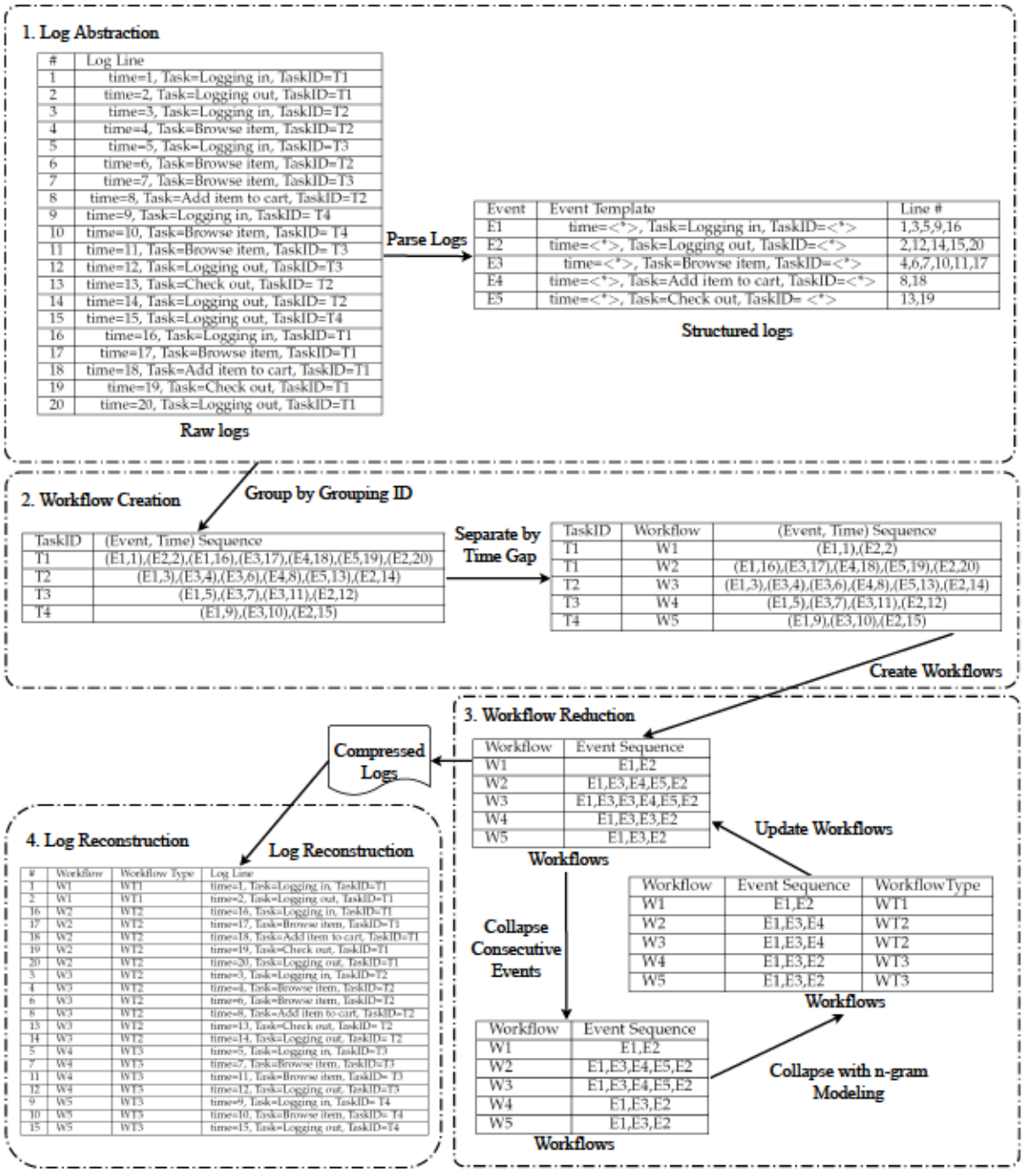

Figure 3.1: The overall flow of our approach *LogAssist* with a running example demonstrating its steps.

lines with an event ID, we are able to use event IDs as the items in our n-gram models in which we compute conditional probabilities.

In this step, *LogAssist* leverages an existing log abstraction tool, *Drain* (P. He et al., 2017), to parse each raw log line into a structured form, i.e., an event template and a list of variables values. We choose to employ *Drain* as it is considered state-of-the-art for log abstraction (Zhu et al., 2019). The default implementation of *Drain* requires one to configure a set of header identifiers (e.g., timestamp and thread ID), which are used by the tool to extract such header information from the execution logs. Accordingly, *LogAssist* also requires one to define the headers for each log dataset. *Drain* parses each raw log line into an event template and a list of variable values (P. He et al., 2017). As demonstrated in Figure 3.1, the event template contains the static information, with a wildcard (i.e., a <*> symbol) in place of all dynamic variables, and a unique event ID for each event type. The list of variable values indicate the dynamic components of the log line. In the running example (Figure 3.1), 20 log lines are abstracted to five types of log events (i.e., E1 through E5). The abstracted log events (i.e., the templates) are used as the basic form for compressing logs in the next steps. Lines 1, 3, 5, 9, and 16 are considered as instances of the same event as they contain a common abstracted template with only differences in the dynamic values (e.g., Timestamp and TaskID). We apply log abstraction to all the logs and assign a unique event ID to every abstracted template.

3.2 Workflow Creation

A sequence of log lines may be related, and together, they may record the process of performing a certain task (T.-H. Chen et al., 2017; Fu et al., 2009; Jiang & Hassan, 2015; Jiang et al., 2008b), e.g., the process of placing an order that includes the sequence of logging in, adding products to the cart, and checking out. Such log sequences (i.e., workflows) provide essential information for practitioners to debug various problems and comprehend the executed user requests (T.-H. Chen et al., 2017; Du et al., 2017; Tan, Pan, Kavulya, Gandhi, & Narasimhan, 2008; Yuan et al., 2010). Hence, in this step, *LogAssist* creates workflows from the parsed log events.

3.2.1 Group log events by grouping ID

As the input logs consist of intermixed events from different workflows, we follow prior work by first grouping the log events by the grouping ID (T.-H. Chen et al., 2017; Fu et al., 2009; Jiang & Hassan, 2015; Jiang et al., 2008b). An example of intermixing events can be seen in Figure 3.1 in the Raw Logs (shown in the first table in Step 1. Log Abstraction) where events of a workflow with TaskID=T2 appearing on lines 3, 4, 6, 8, 13, and 14. Intermixed within these lines are the events of other workflows where TaskID=T3 and TaskID=T4, appearing on lines 5, 7, 11, and 12, and lines 9 and 10, respectively. In practice, this may occur on a much larger scale and two sequential events in a workflow may be separated by tens or possibly hundreds of intermixing log lines. In the running example (Figure 3.1), the grouping ID is "TaskID".

3.2.2 Separate by Time Gap

However, the log events with the same grouping IDs may not necessarily belong to the same workflow, as grouping IDs may be reused by different workflows (e.g., each thread in a thread pool might be reused, so the same thread ID will appear multiple times) (Nageswaran, 1999). Therefore, we further separate the log events with the same grouping ID into separate workflows, based on the time difference between the log events. Our intuition is that log events within the same workflow have smaller time differences while log events from different workflows that reuse the same grouping ID will lead to larger time differences. We use a *find_peaks* algorithm from the signal processing domain (Virtanen et al., 2020) to detect time gaps that separate different workflows. The *find_peaks* algorithm takes an array of data points and finds all local maxima by comparing each data point with its neighbouring points. Specifically, each log line within the group is assigned a time-diff based on the difference between the timestamp of the log line and the timestamp of the previous log line. Then, we use the *find_peaks* algorithm to detect the `peak points` in the time differences. The detected `peak points` are then used to separate the log lines in a group into smaller workflows. In the running example (Figure 3.1), five workflows (i.e., *W1, W2, W3, W4* and *W5*) are created. Two T1 are created since there is a large time gap between their occurrences (line 2 and 16).

3.3 Workflow Reduction

The log events in a workflow may contain redundant information, e.g., repetitive log events and sequences of log events that always appear together (Fu et al., 2013, 2009; Jiang et al., 2008b). Such repetitive log events may mask real problems in the logs or introduce additional challenges in log analysis (J. Chen, Shang, Hassan, Wang, & Lin, 2019; Lin, Zhang, Lou, Zhang, & Chen, 2016; Shang et al., 2013; Xu et al., 2009a). Therefore, *LogAssist* eliminates the redundancies to reduce the workflows into a more concise representation. *LogAssist* performs two steps to reduce the amount of log lines within a workflow: collapsing consecutive events and collapsing with n-gram modeling.

3.3.1 Collapse consecutive events.

LogAssist first reduces the consecutive occurrences of the same event into a single occurrence. Such consecutive occurrences of the same event may be events contained in a loop, or a continuous notification of a process waiting for a resource to become available, which usually indicates repetitive and redundant information (Shang et al., 2013). In the running example (Figure 3.1), both workflows *W3* and *W4* contain two consecutive occurrences of event *E3* as seen in the event sequences *E1,E3,E3,E4,E5,E2* and *E1,E3,E3,E2*. The consecutive occurrences of *E3* are reduced to a single occurrence, resulting in event sequences *E1,E3,E4,E5,E2* and *E1,E3,E2* for workflows *W3* and *W4*, respectively.

3.3.2 Collapse with n-gram modeling.

After collapsing consecutive occurrences of the same events, *LogAssist* further reduces the re-occurring patterns of event sequences into a more concise representation. In addition to collapsing consecutive events as done by Shang et al. (2013), we apply n-gram modeling to further reduce the logs where possible. As we collapse with n-grams with a certainty of 100%, we are able to effectively reduce workflows and subsequently group them into common workflow types while maintaining a high precision of workflow grouping (i.e., ensuring that the workflows in the same group indeed have the same workflow type). For example, if event *E1* is always followed by *E2* and the event sequence *E1,E2* is always followed by *E3*, then the certainty of the event sequence

E1,E2,E3 is 100% given the event *E1*. Thus, we can use *E1* to represent the entire event sequence. Utilizing n-gram to collapse the events allows *LogAssist* to reduce all instances of these workflow types to the same common workflow type representation and group them together. Our intuition is that, if some events always appear in a fixed event sequence, then such an event sequence can be reduced into one event. Specifically, we calculate the conditional probability of a n-gram as:

$$p(e_n|e_1...e_{n-1}) = \frac{count(e_1...e_n)}{count(e_1...e_{n-1}*)} \quad (1)$$

where $(e_1...e_n)$ indicates an event sequence of length n, and $*$ is a wildcard that represents any event. We reduce a n-gram sequence into a single event if the conditional probabilities of the second event through the nth event are all 100% (i.e., $p(e_n|e1...e_{n-1}) = 1$, $p(e_{n-1}|e1...e_{n-2}) = 1$, ..., $p(e_2|e_1) = 1$. Such a reduction guarantees that all the events can be unambiguously represented in the compressed form. We consider 2-grams and 3-grams only, as a prior study by P. He et al. (2018) finds that the repetitiveness of an n-gram in logs starts to become stable when $n \leq 3$. In the running example, the event sequence *E4,E5,E2* always appears together (i.e., the conditional probabilities $p(E2|E4, E5)$ and $p(E5|E4)$ both equal to 1), thus it is reduced into a single event *E4* (i.e., the first event in the sequence) in *W2* and *W3*. This results in the event sequence of *E1,E3,E4,E5,E2* in workflows *W2* and *W3* being reduced to *E1,E3,E4*. Following the collapsing of n-grams, the workflow reduction step once again collapses any consecutive sequences of identical events and applies the n-gram modelling reduction again. This combination of consecutive event and n-gram collapsing repeats as an iterative step until the no further collapsing can be done.

3.4 Log Reconstruction

Finally, the compressed form of logs may need to be reconstructed into the original form to assist with log analysis tasks that need the complete information in the logs. Therefore, *LogAssist* supports log reconstruction that rebuilds the original logs from the compressed form. In particular, our reconstructed logs keep the holistic workflows (i.e., avoiding intermixed log lines across different workflows).

3.5 *LogAssist* is Lossless.

LogAssist provides the ability to view a given workflow in multiple forms at different verbosity levels. While each of these forms is represented by a varying amount of log lines, our approach is lossless as each of these forms can be viewed by expanding and collapsing the workflows where applicable. *LogAssist* contains the complete information of the original log lines (i.e., the corresponding line number in the original form) and allows practitioners to expand the workflows to their original log lines without losing any information. Internally within *LogAssist*, all log lines from the initial raw logs that were passed into the log abstraction step have their line numbers mapped to the resulting reduced workflows. Therefore, *LogAssist* supports reconstructing the original logs based on such line number mappings. No single event is ever permanently lost during log reduction, but rather the events that are hidden in the compressed forms can be accessed by expanding the workflow. In the most reduced form, we represent a workflow as a single log line where the workflow ID label can be used to obtain information on this workflow type. In the most expanded form, we represent the workflow in its entirety showing every single line. In between these forms, there may be a number of other varying representations where inner workflows can be collapsed or expanded, allowing users to choose their desired level of verbosity to suit their own needs, preferences, and tasks.

3.6 An Exemplar Usage Scenario of *LogAssist*

We implemented a web-based graphical user interface as shown in Figure 3.2. The *Workflow Type Details Panel* to the left shows the statistics of a unique workflow type (e.g., the number of workflows that belong to this unique workflow type, the number of events in the unique workflow type, the size of the workflow after compression, and the common log event sequence). The *Workflow Log Report Panel* to the right shows the compressed log lines grouped by their corresponding workflow. By default, we represent each workflow instance as a single line showing the first event in the workflow, its workflow instance ID, and the assigned workflow type ID.

A user may start by looking at the *Workflow Type Details Panel* until they find a workflow type of interest, because the particular workflow is critical to the system behaviour or may be suspected

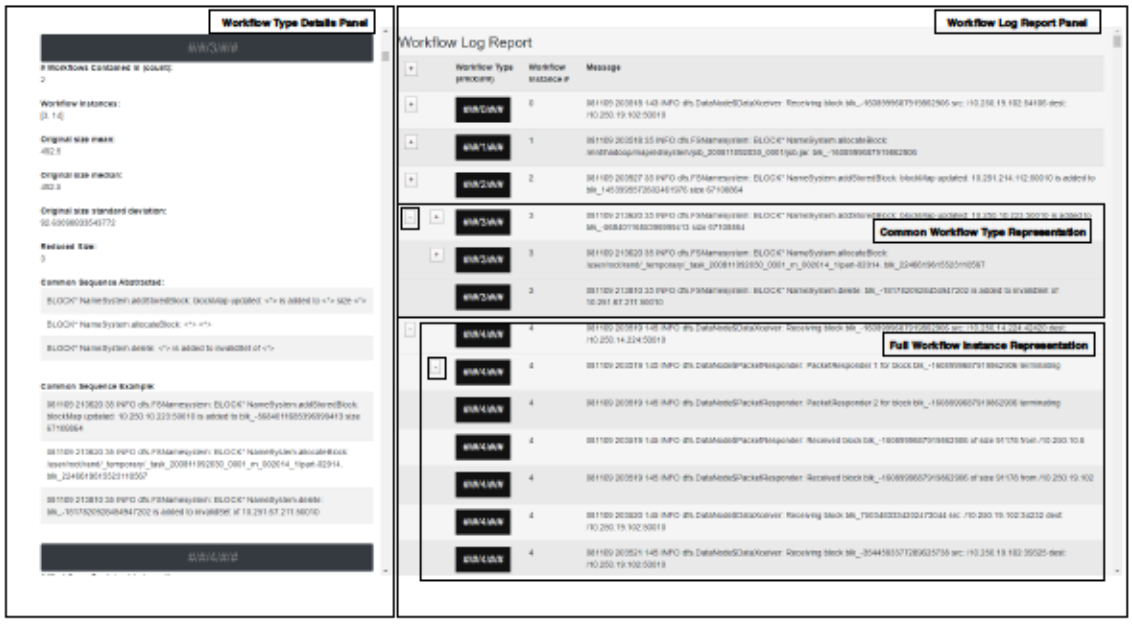

Figure 3.2: An exemplar web-based user interface of *LogAssist*.

of relating to system issues. Then, the user would navigate to the instances of this type and expand the workflow instances in order to gain more details. In the *Workflow Type Details Panel*, users would find various details on the workflows that share this common workflow type, including the abstracted common event sequence and an example workflow instance. The *Workflow Instances* list details all workflow instance IDs of this type, which allows the user to navigate to the workflow instances conveniently. By clicking the "+" button of an instance, as seen in the upper box labeled *Common Workflow Type Representation* in Figure 3.2, the user will expand the workflow instance into the common representation of the workflow type as seen in the *Common Sequence Abstracted* log lines shown in the *Workflow Type Details Panel*. By clicking the inner "+" buttons, users will be able to expand the *Common Workflow Type Representation* further into the *Full Workflow Instance Representation* as seen in the lower box in the *Workflow Log Report Panel* of Figure 3.2. This will reveal log lines of the workflow instance that were abstracted away in the *Common Workflow Type Representation* form, providing the complete details of the workflow to assist the user.

Chapter 4

Evaluation

In this chapter, we evaluate our approach. We select three log datasets to demonstrate the effectiveness of our approach in reducing logs, including two log datasets generated by two open source systems, HDFS and ZooKeeper, and one log dataset generated by one enterprise system (i.e., the Enterprise System, ES). The HDFS and Zookeeper datasets are obtained from a log parsing benchmark (Zhu et al., 2019), while the ES dataset is obtained from our industrial collaborator (Ericsson). We use the thread ID as the grouping ID for the open source systems. Note that, in some distributed systems, logs may contain correlation IDs to correlate logs across nodes/components that are related to the same requests. In such cases, developers may use the correlation ID as the grouping ID when using our approach.

Table 4.1 summarizes our selected log datasets. Due to the non-disclosure agreement, we cannot reveal the detailed information of the logs from ES; however, the logs are large in size, and are generated by a large-scale enterprise system that is used by millions of people around the world on a daily basis. The evaluation of our approach consists of answering three research questions (RQs), which involve a combination of automated analysis and a user study. For each research question, we discuss the motivation, approach, and results.

Table 4.1: A summary of the studied log datasets.

Logging System	Log size	Duration	Grouping ID
HDFS	11M lines	36.68 hours	Thread ID
ZooKeeper	74K lines	62.29 hours	Thread ID
Enterprise System	Very Large	Very Long	Thread ID

4.1 RQ1: How well can logs be compressed into re-occurring event sequences?

Motivation. During the execution, a system often needs to process a large number of re-occurring events (Fu et al., 2009; Jiang et al., 2008b; Xu et al., 2009a). For example, in an e-commerce system, thousands of users may be logging in and logging out on a daily basis. The triggering of such re-occurring events may repeatedly generate the same log event sequences, which may cause wasted efforts and mask important problems captured in logs (Fu et al., 2014; Li et al., 2020). Therefore, we propose *LogAssist* which leverages such re-occurring information to compress raw logs into a conciser form. *LogAssist* first groups the raw logs into workflows, then applies reduction techniques to collapse consecutive events, and finally collapses the events with n-gram modeling. In this RQ, we want to examine how many log lines can be compressed by our approach. If we can compress most of the repeated log event sequences, we may significantly reduce the effort that practitioners need to spend on analyzing the logs.

Approach. We use the following metrics to evaluate the effectiveness of *LogAssist* in compressing the raw logs. For each evaluation metric, we measure its value before and after applying *LogAssist* to compress the raw logs.

- **Number of log lines:** The total number of log lines in the raw logs or in the compressed form.
- **Number of unique workflows:** The number of distinct workflow types that are identified in the raw logs (i.e., before performing workflow reduction) or the number of distinct workflow types remaining in the compressed form (i.e., after performing workflow reduction). The workflows with the same sequence of events in their reduced form are considered to share the same unique workflow type.

- **Workflow size mean:** The average number of log events in a workflow before or after workflow reduction.
- **Workflow size median:** The median number of log events in a workflow before or after workflow reduction.
- **Workflow size st. dev:** The standard deviation of the number of log events in a workflow before or after workflow reduction. A higher standard deviation indicates a high variance of workflow sizes that may cause extra effort in log analysis.

Comparison with prior work. To assist practitioners in identifying deployment problems, Shang et al. (2013) proposed an approach to compare the workflow types between testing and production environments. Although the usage and motivation of the approach is different from *LogAssist*, Shang et al. (2013) also applied workflow reduction. Therefore, we use Shang et al. (2013) as a baseline and compare it with *LogAssist*. Both *LogAssist* and Shang et al. (2013) leverage a dynamic value (e.g., ThreadID or TaskID) to group related events. However, *LogAssist* also applies additional logic for determining event sequences (workflows) where we use the time gap between the events to separate the workflows (i.e., accounting for the reusing of the dynamic values such as ThreadIDs), as explained in Chapter 3.2. Additionally, while *LogAssist* and Shang et al. (2013) both summarize event sequences (workflows) by collapsing consecutive repeating events, Shang et al. (2013) apply this step only once per event sequence (workflow). On the other hand, *LogAssist* applies this step recursively and uses n-gram modeling to further reduce the workflow. This process that combines collapsing consecutive events and collapsing based on n-gram modeling continues iteratively on each workflow until no further reduction can be done.

Shang et al. (2013) group permutations of an event sequence into the same workflow type to reduce the number of unique workflows types. For example, the sequence *E1,E2,E3,E4* and its permutation *E1,E3,E2,E4* are grouped to the same workflow type. As our goal is to assist practitioners with log analysis instead of identifying workflow differences in different deployments, we want to preserve the event orders and do not apply the permutation grouping in our final approach. However, to better compare Shang et al. (2013) with *LogAssist*, we consider with and without permutations for each approach, reporting the reductions in unique workflow types and total log lines.

Table 4.2: The results of applying *LogAssist* to compress the HDFS, Zookeeper, and Enterprise System datasets. *Before* and *After* show the reduction result after applying both consecutive reduction and n-gram (i.e., *Consec.+n-gram*).

	HDFS				Zookeeper				Enterprise System	
	Before	After	Consec. Reduction	Consec. +n-gram	Before	After	Consec. Reduction	Consec. +n-gram	Consec. Reduction	Consec. +n-gram
Number of Log Lines	11,175,579	1,612,315	52.3%	85.6%	74,380	4,543	24.2%	93.9%	22.9%	75.2%
Number of Unique Workflows	72,426	7,372	43.4%	89.8%	329	98	42.9%	70.2%	3.1%	3.1%
Workflow Size Mean	21.2	3.1	52.3%	85.6%	26.0	1.6	24.2%	93.9%	22.3%	75.2%
Workflow Size St. Dev	1,019.1	63.5	89.1%	93.8%	534.7	0.88	2.4%	99.8%	22.6%	75.4%
Workflow Size Median	3	2	0%	33.3%	3	2	33.3%	33.3%	0%	50.0%

Evaluating the effect of n-gram modeling. Prior work (Shang et al., 2013) collapses consecutive repeating events during workflow creation but does not use n-gram modeling. In order to understand the effect of applying n-gram modeling for further reducing the log lines, we compare *LogAssist* with its simplified version that does not apply the "collapse with n-gram modeling" step. Specifically, the simplified version does a single pass of "collapse consecutive events" instead of applying the combined "collapse consecutive events" and "collapse with n-gram modeling" steps in an iterative manner (as done in *LogAssist*).

Results. ***LogAssist* compresses the raw logs into a concise representation that is 75.2% to 93.9% smaller.** Table 4.2 shows the results of measuring the evaluation metrics on the raw logs (i.e., before applying *LogAssist*) and on the compressed representation (i.e., after applying *LogAssist*). Our results show that *LogAssist* can compress a significant amount of log lines in the studied systems: 85.6%, 93.9%, and 75.2% for HDFS, Zookeeper, and Enterprise System, respectively. Our results indicate that there are many re-occurring log events or event sequences that practitioners may be able to skip during log analysis.

***LogAssist* reduces the unique workflow types by up to 89.8%.** The unique workflow types indicate the complexity of the system behavior recorded in the logs. The larger the number of unique workflow types, the more diverse the system behavior, thus more effort may be needed to analyze the system behavior. As shown in Table 4.2, the unique workflow types are reduced by 70.2% to 89.8% for the open source systems. The results show that a unique workflow type may have different variances that can be identified by *LogAssist*. In other words, *LogAssist* may help practitioners reduce the needed effort to navigate and study the sequences of log events and the dynamic execution paths using the compressed workflows (see our user study in RQ3). The unique workflow types are only reduced by 3.1% for ES. Although we cannot disclose the details for ES, we find that

Table 4.3: The number of workflows for which the log events are compressed. The numbers in the parentheses show the percentage.

	Total workflows	Num. of workflows compressed
HDFS	527,326	334,752 (63.5%)
Zookeeper	2,857	2,787 (97.6%)
Enterprise System	–	– (88.1%)

the smaller reduction in the number of unique workflow types is due to the nature of the analyzed workflows i.e., each workflow type of ES has fairly fixed event sequences (i.e., with less variance). However, our approach can still compress most of the re-occurring log lines in ES.

***LogAssist* reduces the average size of a workflow by 75.2% to 93.9%.** Table 4.3 shows the number of workflows where the logs are compressed. We find that most workflows can be compressed: 63.5%, 97.6%, and 88.1% of the workflows are compressed in HDFS, Zookeeper, and ES, respectively. Table 4.2 also shows the statistics of the number of log lines in each workflow. On average, *LogAssist* reduces the size of each workflow by 75.2% to 93.9%. Taking the HDFS logs for example, the average number of log events in each workflow is reduced from 21 to less than 3. In addition, the standard deviation of the number of log events in a workflow is also significantly reduced (75.4% to 99.8%), meaning that the workflow sizes become more consistent after applying *LogAssist*. Our findings show that there is a high-level of repetition of log events within a workflow. The reduction in the median workflow size is smaller, which is due to the fact that most of the workflows are small in size (e.g., the median workflow size is three log events for the two studied open source systems even before compression). Additionally, for each system we perform a Wilcoxon signed-rank test to compare the sizes of the original workflows and the reduced workflows. Our results indicate that *LogAssist* can provide a statistically significant reduction in the size of workflows in logs with a value of $p<0.001$ across all three systems.

***LogAssist* is more effective in reducing the log events for larger workflows which are more likely to contain repetitive information.** Table 4.4 shows the percentage reduction for workflows with a size less than, equal to, and greater than the median workflow size. In all three systems, workflows with sizes greater than the median show a significantly higher reduction percentage (65.90% to 85.18%) than those that are less than or equal to the median size (14.83% to 41.07%). The results show that larger workflows are more likely to be reduced compared to smaller ones. Larger

Table 4.4: Reduction % based on size of workflow compared to the median workflow size.

	HDFS	Zookeeper	Enterprise System
<Median	14.83	46.43	N/A
Median	19.01	37.37	41.07
>Median	65.90	85.18	69.82

workflows may contain more repetition, which results in higher reduction rates. Additionally, when using a threshold of 100% probability for the n-gram collapsing, the opportunity to reduce these logs is highly dependent on the nature of the workflows. If the events do not follow any specific ordered sequence, the n-gram probabilities may not meet the required threshold and subsequently n-gram reduction will not be possible.

Application of n-gram modeling in *LogAssist* is significantly more effective than applying consecutive collapsing of duplicate events alone. As shown in Table 4.2, applying both n-gram collapsing and consecutive collapsing of duplicate events shows significantly higher reductions compared to applying only consecutive collapsing. By applying n-gram, we see 33.3% to 69.7% *additional* reduction in the number of log lines in all studied systems, and 27.3% to 46.5% in the number of unique workflows in HDFS and Zookeeper. The mean, median, and standard deviation of workflow sizes show additional reductions of 33.3% to 69.7%, 4.7% to 97.4%, and 33.3% to 50%, respectively, across all three systems.

***LogAssist* outperforms current state-of-the-art in grouping common events and reducing total log lines.** Table 4.5 shows that both *LogAssist* and its variation with permutation grouping outperform Shang et al. (2013). As previously stated, due to differing goals between *LogAssist* and Shang et al. (2013), we do not apply permutation grouping in our final approach as we aim to keep the distinction between different orders of the event sequences in the workflows. *LogAssist* can be extended to include this functionality if required. However, to ease the comparison between the two approaches, we also included grouping by permutation in *LogAssist*. Table 4.5 shows the comparison results. The findings indicate that in all cases, *LogAssist* outperforms Shang et al. (2013) for both the percentage reduction in unique workflow types and log lines. Comparing both approaches without grouping by permutations shows an additional 27.35% to 46.4% reduction in unique workflow types for HDFS and Zookeeper when using *LogAssist*. Comparing both approaches with grouping by permutations shows an additional 8.35% to 26.14% reduction in unique

Table 4.5: A comparison between *LogAssist* and current state-of-the-art approach by Shang et al. (2013) for reduction % in unique workflow types (with and without permutations), and reduction % in total log lines.

	Reduction % in Unique Workflow Types				Reduction % in Log Lines	
	w/ permutations		w/o permutations			
	LogAssist	Shang et al., ICSE2013	*LogAssist*	Shang et al., ICSE2013	*LogAssist*	Shang et al., ICSE2013
HDFS	95.03	86.68	89.80	43.40	85.60	52.30
Zookeeper	72.64	46.50	70.20	42.85	93.90	24.20
Enterprise System	3.10	3.10	3.10	3.10	75.20	22.90

workflow types for HDFS and Zookeeper when using *LogAssist*. Finally, comparing Shang et al. (2013) with permutation grouping to the default form of *LogAssist* without permutation grouping, *LogAssist* still shows an additional 3.12% to 23.7% percent reduction in unique workflow types. Both approaches have the same reduction (3.1%) in the unique workflow types in the Enterprise system. However, the results show that *LogAssist* achieves an additional 33.3% to 69.7% reduction in total log lines over Shang et al. (2013). The reason is that Shang et al. (2013) only reduce individual workflows by collapsing consecutive duplicate events. On the other hand, *LogAssist* applies an iterative approach which includes collapsing consecutive duplicate events in combination with collapsing using n-gram modeling.

4.2 RQ2: How much can *LogAssist* reduce the volume of logs needed to be examined in log analysis tasks?

Motivation. Due to the sheer size of logs, practitioners often search for keywords such as *"error"* or *"exception"* to first locate potential problems that occurred during in-house tests or regular user usage (T.-H. Chen et al., 2017; Jiang & Hassan, 2015; Shang et al., 2013). After locating the problematic log lines containing the keywords, practitioners then need to analyze the potential root cause by manually studying the related log lines. For example, practitioners need to manually identify which log event sequences led to the exception (LaToza & Myers, 2010; Nagappan, Wu, & Vouk, 2009; Tan et al., 2008). This log analysis process can be very time-consuming, since there may be thousands of log lines that contain the keywords. *LogAssist* groups logs into workflows and compresses the logs by identifying common log event sequences. The unique workflows that

Table 4.6: Keywords for certain log analysis tasks for each studied system.

		Keywords*	Rationale
HDFS	K1-Normal	served block	The keywords are related to data block being written to or read. The keywords can be used to estimate the load of the system.
	K2-Issue	unexpected error trying to delete block	The keywords are related to a reported bug in HDFS on disk.[1]
	K3-Issue	redundant addStored-Block request received for	The keywords correspond to a warning that may indicate data loss.[2]
Zookeeper	K1-Normal	accepted socket connection from	The keywords are related to connection being established with the Zookeeper server. The keywords are used to estimate system behaviours under load, such as how long a connection lasts.
	K2-Issue	unexpected exception causing shutdown	The keywords indicate a common exception that may happen during data transmission issues.[3]
	K3-Issue	caught end of stream exception	The keywords indicate a common exception in Zookeeper related to data storage and snapshot management.[4]

* Note: The entire phrases are used as keywords to search.
[1] https://issues.apache.org/jira/browse/HDFS-4544
[2] https://news.ycombinator.com/item?id=9476515
[3] https://mapr.com/support/s/article/Zookeeper-Unexpected-exception-causing-shutdown-while-sock-still-open-java-io-IOException-Unreasonable-length?language=en_US
[4] https://stackoverflow.com/questions/38887977/zookeeper-keeps-getting-endofstreamexception-causing-a-crash

LogAssist identifies may help reduce the amount of logs that practitioners need to go through when searching and debugging for problematic log lines. Therefore, in this RQ, we study how many log lines may need to be examined given various keywords before and after applying *LogAssist*.

Approach. We follow prior work (Shang et al., 2013) to study how effectively *LogAssist* can reduce the volume of logs to be examined in log analysis tasks. We perform several typical log analysis tasks on the raw logs and on the compressed representations. We then determine the number of log lines that would need to be examined before and after applying *LogAssist*, respectively. On each log dataset, we search for a keyword in the logs and examine the searched logs, which is commonly done in log analysis practices (ElasticSearch, n.d.; A. Oliner et al., 2012; Splunk, 2017). We consider three tasks: one task for searching and analyzing a normal message, and two tasks for searching and analyzing certain system runtime issues (e.g., warnings, errors, or exceptions). To identify the keywords, we manually examine the logs and uncover the log events that are related to normal messages and system runtime issues. Then, we choose the keywords in the most frequently appearing log event for each of the three categories, since those events are the ones that practitioners may need to spend the most time examining (Shang et al., 2013). We list and explain the keywords that we use to search for log lines in each of the studied systems in Table 4.6.

For each task, we evaluate the number of examined log lines based on two scenarios:

- **Scenario 1: Examining only the searched log lines.** For some searched log lines, the log line itself may contain all required information. In this scenario, we assume that practitioners only examine the log lines that match with the keywords.

- **Scenario 2: Examining the entire workflow that contains the searched log lines.** However, for some searched log lines, other log lines related to the searched ones may also need to be examined (e.g., logs in the same execution sequence) (LaToza & Myers, 2010; Tan et al., 2008; Yuan et al., 2010). Therefore, in this scenario, we assume that practitioners examine all the log lines related to the searched log lines (i.e., all log lines in the workflows containing the searched keywords).

Under each scenario, we evaluate the number of examined log lines using two representations of the logs:

- **Original logs.** Examining the searched log lines (and related log lines in the case of scenario 2) in the original raw logs.

- **Compressed form (unique workflows).** Examining the searched log lines (and related log lines in the case of scenario 2) in the compressed form, considering only each unique workflow type once. In the compressed form, we consider only a single instance of each distinct workflow type, since workflows of the same distinct type share a common compressed form.

Results. ***LogAssist*** **reduces the number of searched log lines that need to be examined by practitioners by 75% to 99%.** Table 4.7 compares the number of log lines to be examined using different representations of the logs (i.e., the original and the compressed forms), assuming that practitioners only examine the searched log lines. We find that without *LogAssist*, keyword search returns up to 428K log lines for the normal message, which is impossible to manually inspect. Even when searching for log lines that indicate system runtime issues, keyword search returns several hundreds or thousands of log lines. After applying *LogAssist*, the log lines to examine are greatly reduced, with the log lines containing the searched keyword only appearing in a small subset of the workflows. Compared to using the original logs, using *LogAssist* can reduce the number of log lines that need to be inspected by up to 99%.

Table 4.7: Number of log lines to be examined using different representation of logs (Scenario 1: examining only the searched log lines).

Search key	HFDS			Zookeeper			Enterprise System
	Original logs	Compressed form	Reduction	Original logs	Compressed form	Reduction	Reduction
K1-Normal	428,726	803	99.81%	2,020	52	97.43%	75.00%
K2-Issue	5,545	25	99.55%	590	4	99.32%	80.00%
K3-Issue	975	96	90.15%	1,670	45	97.31%	75.00%

Table 4.8: Number of log lines to be examined using different representation of logs (Scenario 2: examining the entire workflows that contain the searched log lines).

Search key	HFDS			Zookeeper			Enterprise System
	Original logs	Compressed form	Reduction	Original logs	Compressed form	Reduction	Reduction
K1-Normal	861,998	10,153	98.82%	80,37	907	88.71%	75.00%
K2-Issue	1,375,884	2,964	99.78%	1,190	7	99.41%	77.78%
K3-Issue	3,257,875	284,926	90.15%	8,477	803	90.53%	75.00%

***LogAssist* dramatically compresses the searched-line-related workflows that need to be examined by practitioners (i.e., by up to 99% reduction).** Table 4.8 compares the number of log lines to be examined using different representations of the logs, assuming that practitioners need to examine the entire workflows containing the searched log lines (which is a common practice in log analysis and debugging (LaToza & Myers, 2010; Tan et al., 2008; Yuan et al., 2010)). We find that the number of lines that need to be examined in the raw logs increased significantly to up to millions. After using *LogAssist* to compress the log lines, we can reduce the number of log lines that need to be examined by 75% to 99%. Although the reduction is large, we find that sometimes practitioners may still need to investigate several thousands of log lines. After some investigation, we find that it is because many of the log events that contain the search keywords are generated by different log event sequences (i.e., different workflows). Namely, there may be different causes that lead to a normal message or an issue-indicating message. In addition, some workflows may contain hundreds of log events, which increases the number of log lines that need to be examined. However, our results can still help practitioners identify the unique workflows that need to be examined and assist them in examining the event sequences in the workflows.

Table 4.9 shows the number/percentage of workflows and workflow types in which the keywords appear. We exclude the raw numbers for ES due to the NDA. The percentage of workflows

Table 4.9: The number of workflows and workflow types in which the search keys appear.

Search key	HFDS		Zookeeper		Enterprise System	
	Workflows (%)	Workflow Types (%)	Workflows (%)	Workflow Types (%)	Workflows (%)	Workflow Types (%)
K1-Normal	126,873 (24.06%)	475 (6.44%)	129 (4.52%)	18 (18.37%)	— (14.29%)	— (9.68%)
K2-Issue	29 (0.00549%)	23 (0.3119%)	590 (20.65%)	1 (1.02%)	— (4.67%)	— (4.67%)
K3-Issue	100 (0.0189%)	93 (1.262%)	161 (5.64%)	17 (17.35%)	— (6.45%)	— (6.45%)

that contain the keywords range from 0.00549% to 24.06%, 4.52% to 20.54% and 4.67% to 14.29% for HDFS, Zookeeper, and ES, respectively. The percentage of workflow types that contain the keywords range from 0.3119% to 6.44%, 1.02% to 18.37%, and 4.67% to 9.68% for HDFS, Zookeeper, and ES, respectively. The results show no significant correlation between the reduction percentages shown in Table 4.7 and Table 4.8, and the number of workflows and workflow types that contain these keywords.

4.3 RQ3: How much can *LogAssist* help improve users' log analysis experiences?

Motivation. Our first two research questions seek to quantitatively study the effectiveness of *LogAssist* for compressing logs and assisting with log analysis. In this research question, we aim to qualitatively evaluate how well *LogAssist* can assist practitioners in performing log analysis tasks and reduce the needed efforts. Therefore, we perform a user study in which we invite practitioners and researchers to perform typical log analysis tasks using *LogAssist*. We compare the user study results with and without using the tool.

Approach. We performed a user study with 19 participants, among whom 7 are software engineering practitioners and the other 12 are software engineering researchers (e.g., graduate students). We asked the participants to perform six log analysis tasks on the Zookeeper and HDFS datasets. The tasks and the datasets are publicly available online[1]. *LogAssist* uses a concise log representation to assist users in log analysis while still providing users the flexibility to access the entire information in the logs. Therefore, we design tasks that require users to obtain information from both the concise representation of the logs and the logs that are hidden from the concise representation.

For the purpose of the user study, we provide a subset of the each of the log datasets for the

[1]https://github.com/SteveLocke/LogAssist-Artifacts.git

HDFS and Zookeeper systems. As we ask participants to record the time taken to complete tasks, we intentionally provide a relatively smaller sample of the datasets to ensure that participants and their varying device specifications can all support the log sizes with similar performance. The HDFS dataset sample is 5,095KB in size and consists of 37,002 log lines, while the Zookeeper dataset sample is 3,244KB in size and consists of 25,000 log lines. While these samples are significantly smaller than the complete datasets, each sample still contains a large number of log lines, sufficiently reflecting the challenge related to large log size, as manually analysis on such sizes remains quite difficult.

As even the most complex tasks are composed of smaller tasks, we chose to select a set of smaller tasks in the user study and provide specific instruction in order to ensure that participants of varying backgrounds could complete the tasks within a reasonable amount of time. Our designed tasks covered a variety of typical log analysis tasks including analyzing the event sequence that leads to an error, counting the occurrences of certain event sequences (i.e., workflows), counting the occurrences of certain operations that encounter errors, and summarizing key information (e.g., the opened channels) in the logs. For example, one user study task involves determining the count of an ordered pair of events which occur together as part of the same event sequence. Participants are given instructions on how to use *LogAssist*, a starting point in the logs, and description of the event pairs to be found. In practice, this task will likely be part of a more complex task requiring additional analysis on the workflow.

Each participant was required to use *LogAssist* in three tasks and avoid using the tool (i.e., using only the raw logs) in the other three tasks. Each participant was given a randomized and evenly distributed assignment for which three tasks that they have access to *LogAssist*. For each task performed, we asked the participant to record the time spent on the task, and their results of performing the task. We also asked the participants to evaluate whether *LogAssist* improves their experience of performing the tasks over using only the raw logs, using a scale of 1 (strongly disagree) to 5 (strongly agree). Users were given the option of including additional qualitative feedback in the form of unstructured comments. Every task is designed to be able to be completed with or without using *LogAssist*. In practice, sometimes the required information may not be readily available in a workflow's compressed form. Thus, we design three out of the six tasks (i.e., T1, T2,

Table 4.10: The average time with, and without *LogAssist* and the % reduction. The time values are represented in minutes for each individual task, as well as the total for all tasks combined.

	Avg. time w/o. *LogAssist* (min)	Avg. time w. *LogAssist* (min)	Time Improvement (%)
T1	13.65	3.35	75.46
T2	8.26	5.32	35.59
T3	3.99	4.59	-15.04
T4	6.565	3.98	39.38
T5	2.85	5.31	-86.32
T6	5.56	0.95	82.91
Total	40.88	23.51	42.49

and T3) to require expanding workflows from their compressed forms when using *LogAssist*.

Results. **On average, *LogAssist* reduces the amount of time needed for the participants to perform the log analysis tasks by 42%.** Table 4.10 compares, for each task, the average time needed for the participants to perform the task with and without *LogAssist*. In four out of the six tasks, the time required to perform the task was reduced by 35.59% to 82.91% with *LogAssist*. Our results also show that the tasks that require expanding the workflows do not affect the effectiveness of *LogAssist*, as *LogAssist* can still reduce the time needed for performing tasks that require such expansion (e.g., T1 and T2). However, in two of the six tasks, the required time was increased by 15.04% to 86.32% with *LogAssist*. These two tasks are the simplest tasks (i.e., the participants took the shortest time to perform these two tasks without using *LogAssist*), for which *LogAssist* could not further simplify. While *LogAssist* is able to reduce the amount of time needed for log analysis tasks, there is also an inherent learning curve that the participants experience when using a new tool for the first time. In simpler and shorter tasks, this overhead may become more apparent and possibly increase the overall task time. Nevertheless, using *LogAssist* helped the users to significantly reduce the total needed time to perform all the assigned tasks by 42.49%.

For each task, we also perform a Wilcoxon rank-sum test to compare the time taken by the participants to complete the task with and without the assistance of *LogAssist*. Due to the small sample size, only two of the six tasks (T1 and T6) show a statistically significant reduction in completion time when using *LogAssist*. However, the result shows a statistically significant reduction in the overall completion time of the tasks when using *LogAssist* ($p<0.01$).

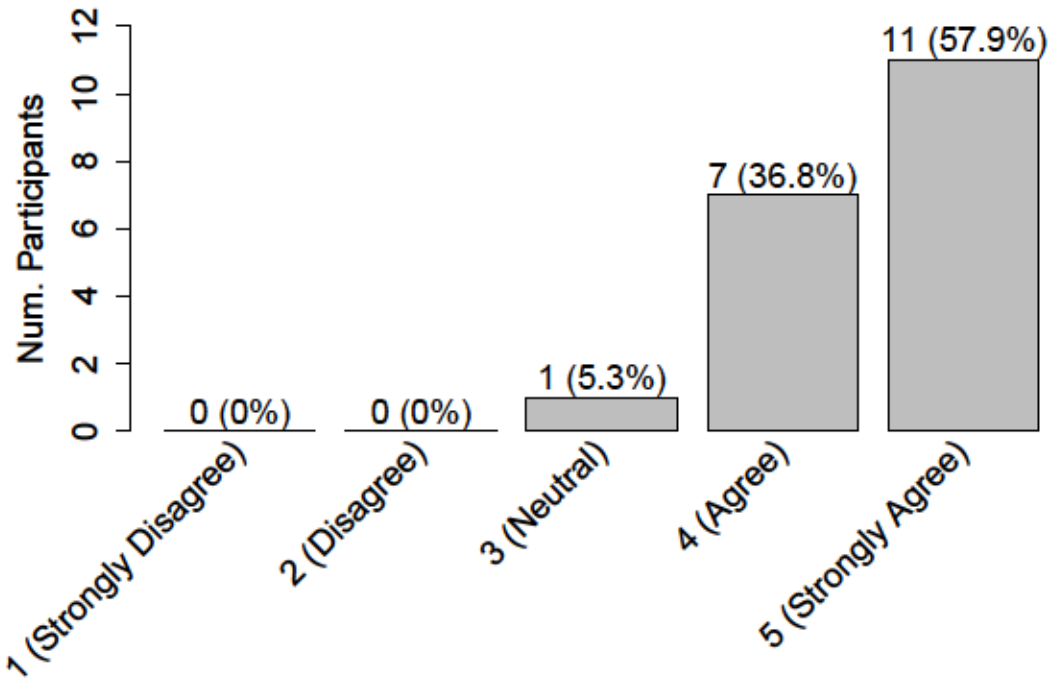

Figure 4.1: User provided rating for the usefulness of *LogAssist*.

***LogAssist* improves the users' experience of performing the log analysis tasks.** As shown in Figure 4.1, 18 out of 19 (94.7%) participants agreed or strongly agreed that *LogAssist* effectively improves their log analysis experience, while only one participant had a neutral opinion on the helpfulness of the tool. On average, participants assigned *LogAssist* a rating of 4.53 out of 5. After speaking with the participant who had neutral opinion, the participant indicated that she rated the tool as such due to experiencing some frustration while performing one task. She assumed that the task should be simple, but instead found the task challenging even with the tool, leading her to assume that she may not have been using the tool in an optimal fashion. Overall, as *LogAssist* extracts meaningful workflows from the raw logs and abstracts the workflows into a concise set of common event sequences (i.e., unique workflow types), *LogAssist* can effectively simplify users' log analysis tasks.

Generally, the participants found *LogAssist* to be helpful and provide benefits over using simply the raw logs. Many expressed their appreciation of the tool and its capabilities including automated workflow extraction, insights, visualizations, and the ability to perform some tasks much more quickly. Some comments by participants seemed to indicate that they felt that developers needed

to be familiar with the concept of a workflow and be familiar with how to use the tool in order to get the most from *LogAssist*. Similarly, one participant felt it would be helpful to further highlight the underlying logic behind *LogAssist* to help users better understand how to operate it. While we did provide participants with documentation outlining explanations of workflows and instructions on how to utilize *LogAssist*, we recognize that there is a learning curve not only with *LogAssist*, but with log analysis in general. As our study included participants from varying levels of experience in log analysis, we expect a similar variation in the learning curve experienced. We expect that future practitioners who adopt *LogAssist* will experience a relatively small learning curve based on their domain knowledge.

Despite positive feedback outlining the benefits of *LogAssist*, some participants did suggest some additional features and improvements that they felt could benefit *LogAssist*. These suggested features and improvements included additional filtering and sorting options, and bi-directional quick navigation from statistics to workflows. These suggestions did not highlight an inability to perform specific tasks, but rather, possible ways to further improve the speed of performing tasks when using *LogAssist*, and options to allow users to customize their interface and experience.

Chapter 5

Lessons Learned

Logs are very repetitive, while most of the log information can be compressed without impacting the usefulness of logs. Prior research (Hassan, Martin, Flora, Mansfield, & Dietz, 2008; Liu et al., 2019; Yao, Li, et al., 2020) studies approaches for compressing log data. However, such log compression approaches usually compress logs into a form that cannot be analyzed directly (i.e., in a encoded format). In this work, we propose an approach that compresses logs into a concise form that enables practitioners to conduct log analysis effectively. Besides, practitioners can expand detailed log information when needed, which ensures that practitioners can always find the information that they are interested in, in a more efficient manner.

Re-organizing logs into meaningful workflows can improve practitioners' experience of log analysis. Logs are typically recorded in log files based on when they are generated during the execution of systems. While log files keep the time-based order of the log lines, it is difficult for practitioners to examine the logs, as the log lines of one workflow (e.g., the transaction of checking out a product) are usually intermixed with the lines of other workflows (e.g., ordering supplies or browsing). Our approach leverages the grouping ID information, which is usually available in system logs, to separate the log lines of different workflows. Hence, practitioners can focus on a particular workflow that they are interested in when conducting log analysis (e.g., when diagnosing the cause of an error).

N-gram models can effectively capture the re-occurring patterns in the workflows. Software logs are repetitive, not only in the repetition of the same events, but also in the repetition of the

log sequences (Jiang et al., 2008b; Shang et al., 2013; Xu et al., 2009a). Prior work uses n-gram models to measure the repetitiveness of log data (Yao, Li, et al., 2020) or to identify the static parts of a log line (Dai, Li, Shang, Chen, & Chen, 2020). In this work, we find that using n-gram models (after grouping workflows) can effectively capture such repetition of log sequences and allow us to leverage the captured repetition to further compress the logs into a concise form for log analysis. While our study consisted of only reducing an n-gram sequence into a single event if the conditional probabilities of the second event through the nth event are all 100% (i.e., $p(e_n|e1...e_{n-1}) = 1$, this probability is a hyper-parameter that can be explored in future work. The effects of a threshold analysis which relaxes this probability value would likely open the possibility for further grouping between similar workflow types, but with the added risk of grouping workflows that may be perceived as distinctly different workflow types.

Better tools and support (e.g., a log IDE) are important for practitioners to improve their experience and effectiveness of log analysis. Existing log analysis tools (e.g., Splunk or Elastic) usually support effective log search using keywords. However, such tools do not help practitioners analyze the searched log lines in a more organized fashion (e.g., workflow or recurring log patterns). In this work, we propose a log IDE, to allow practitioners to search all the information they need while only presenting a concise form of information for practitioners to analyze. As indicated by our user study, such an IDE can significantly improve practitioners' experience of performing log analysis tasks. Future research on log analysis should aim to assist practitioners using similar tools.

Providing a concise representation of logs while still providing practitioners the flexibility to access the complete information in the logs. *LogAssist* compresses the logs into a concise form that may simplify practitioners' log analysis tasks. However, practitioners may need to access some detailed log information that is hidden from the concise form. Therefore, *LogAssist* also enables practitioners to search and expand all the information in the original logs. By providing the ability to view a given workflow in multiple forms and at different verbosity levels, *LogAssist* provides a lossless reduction that is flexible. A practitioner may expand or collapse any given workflow to suit their own needs, preferences, and tasks as they see fit, without losing any information from the original logs. Our user study demonstrates that such a combination can effectively improve practitioners' log analysis experiences.

Chapter 6

Threats to Validity

In this chapter, we discuss the threats to validity of our study.

6.1 External validity.

We conducted our experiment on logs from one enterprise and two open source systems. Although the log datasets that we use are from large-scale systems in different domains and are widely used in prior studies (Zhu et al., 2019), our results may not be generalized to other systems. Future studies are needed to verify the effectiveness of our approach on other systems.

6.2 Construct validity.

We evaluate our approach by following a prior study (Shang et al., 2013). Namely, we identify keywords that are related to the most common errors, exceptions, and normal messages. We then use the keywords to evaluate how much effort we can reduce when inspecting the search results. However, the results may not truly represent how much effort is reduced. To mitigate the threat, we conduct a user study in RQ3 to further evaluate the effectiveness of *LogAssist*. In our user study, we use time to measure the effectiveness of *LogAssist* in assisting practitioners with log analysis. There may be other metrics that may be used such as the success rate of finishing the task correctly. Nevertheless, we find that *LogAssist* can also help users finish the log analysis tasks with a much higher success rate (i.e., 60% higher than without *LogAssist*).

In our user study, rather than providing participants with long and complex tasks, we designed the study to include several smaller tasks in order to ensure that participants of varying backgrounds could complete the tasks within a relatively short time-frame. Other possible reasons for the relatively short completion time may include participants guessing, giving up, or believing they have completed a task prematurely. With respect to the complexity of the tasks, even the most complex of tasks are composed of smaller tasks. Furthermore, a non-complex task may contain many repetitive simple tasks that collectively become a time-consuming task. As the logs used in this thesis are real-world logs, we consider the associated tasks to be real-world tasks, and do not consider the time requirement of the tasks to directly correlate with the complexity.

In our workload creation step (Chapter 3.2), we leverage a popular algorithm in the signal processing field to identify gaps between workflows. Although through our manual investigation and the user study, we did not find workflows that are incorrectly identified, future studies are encouraged to compare different algorithms for identifying gaps between workflows.

Chapter 7

Related Work

In this chapter, we discuss related work in three areas: log analysis, understanding system workflows, and log compression.

7.1 Log analysis.

Many prior studies focus on using logs to assist in debugging and understanding system execution. A common log analysis approach is to group the log lines using grouping IDs, and then apply machine learning techniques to detect anomalies (T.-H. Chen et al., 2017; Du et al., 2017; Jiang et al., 2008b; Syer et al., 2013, 2014; Xu et al., 2009a). Such anomalies may be an indication of the problem that happened during system execution. For example, Xu et al. (2009a) propose an approach to first group log lines using grouping ID and then apply principal component analysis to detect anomalies. Jiang et al. (2008b) group log lines using grouping ID and apply z-stat to detect anomalies. Syer et al. (2013, 2014) use hierarchical clustering to identify anomalies in execution logs. Du et al. (2017) leverage deep learning models (i.e., LSTM) to detect anomalies in log sequences. T.-H. Chen et al. (2017) discuss a decade of experience on applying machine learning techniques to analyze logs to assist load test analysis. In this work, we also use grouping IDs to separate log lines into workflows. However, our goal is not only to detect anomalies, but also to help practitioners understand and navigate system execution information.

7.2 Understanding system workflows.

Many prior studies try to assist practitioners in understanding system workflows (e.g., event sequences) to assist in debugging and test design. Yuan et al. (2010) analyze log lines to uncover system execution paths in the source code. Tan et al. (2008) analyze log lines by using state machines to model system execution. T.-H. Chen, Shang, Hassan, Nasser, and Flora (2016) leverage log lines to analyze system workflows and recommend where to place caches. J. Chen et al. (2019) propose approaches to extract representative workflows from production logs to assist with load test design at different levels of granularity. Lin et al. (2016) use clustering to identify similar workflows in logs to assist with workflow comprehension. Workflow understanding is also very popular in the software industry. Commercial tools such as Elastic (n.d.) allow practitioners to search log lines using keywords, and provide different charts (i.e., dashboard) to visualize the matched log lines. Different from prior studies, *LogAssist* aims to provide a more structured representation for log lines. *LogAssist* helps reduce the amount of information that practitioners need to investigate, and can assist in log analysis tasks.

The closest work to ours is by Shang et al. (2013). While Shang et al. (2013) seek to solve the issue of finding deployment bugs in big data applications, *LogAssist* seeks to summarize logs into workflows to facilitate log analysis tasks. The difference in the goals leads to different techniques (as described in Chapter 4.2) and their provided benefits. As discussed in Chapter 4.2, *LogAssist* provides a more concise form of logs than Shang et al. (2013). Furthermore, *LogAssist* allows for transforming the logs into a more readable and comprehensible format where intermixed logs are grouped into relevant workflows. The transformed logs also provide various representations by expanding/collapsing portions of the workflow that allow for even fewer lines to scroll through. *LogAssist* also provides statistics on workflows and workflow types (such as their frequency, workflows sharing the same common workflow type, and the information about the static and dynamic components of the log events in the workflows).

7.3 Log compression.

Prior work (Balakrishnan & Sahoo, 2006; Christensen & Li, 2013; Feng, Wu, & Li, 2016; Hätönen, Boulicaut, Klemettinen, Miettinen, & Masson, 2003; Liu et al., 2019; Mell & Harang, 2014; Otten, 2008; Skibiński & Swacha, 2007) proposes approaches for compressing log files. These approaches usually compress logs through log transformation or text replacement. Some research considers transforming existing log lines in a way to improve the size of the compressed logs. This line of research leverages two main approaches for such a transformation, namely log clustering (Christensen & Li, 2013; Feng et al., 2016) and log transposing (Mell & Harang, 2014). Prior work also compresses logs by replacing long and repetitive text in log files with shorter representations (Balakrishnan & Sahoo, 2006; Hätönen et al., 2003; Liu et al., 2019; Otten, 2008; Skibiński & Swacha, 2007). For example, Otten (2008) transforms all timestamps and IP addresses in a log file to binary representations, then replace the static tokens in log files (i.e., static words and phrases) with shorter representations. Recently, Liu et al. (2019) propose a log preprocessing approach (i.e., *Logzip*) that extracts log templates from log data and replaces each template with a shorter representation (e.g., a unique ID). Yao, Li, et al. (2020) evaluate the performance of various general compression algorithms on log compression. They find that logs are highly repetitive and highlight the difference between compressing logs and natural language text. These approaches transform logs into a compressed form that does not allow directly performing log analysis without decompression. In this work, we propose an approach to compress logs into a concise form while allowing practitioners to access the complete information in the logs on demand, without a decompression process.

Chapter 8

Conclusion

In this thesis, we present *LogAssist*, a novel approach for assisting practitioners with log analysis. *LogAssist* successfully identifies common workflow types by condensing extracted workflows using consecutive event sequences and n-gram models. In particular, by evaluating *LogAssist* on one enterprise and two open source systems, we find that *LogAssist* is able to significantly reduce the amount of log lines that need to be examined in typical log analysis tasks and the associated effort. In particular, this thesis makes the following contributions:

- We propose a novel approach that can effectively compress raw logs into concise forms which can simplify and facilitate practitioners' log analysis tasks.
- We demonstrate the importance of untangling the intermixing events contained in raw logs into meaningful event sequences (i.e, workflows) and apply statistical techniques (e.g., n-gram models) to identify such re-occurring patterns of event sequences.
- We share the lessons that we have learned while developing and adopting our approach, which can provide valuable insights for researchers and practitioners wishing to develop or adopt similar tools to assist with log analysis tasks.

References

Automated root cause analysis for spark application failures - o'reilly media. (2017). ((Last accessed August 13, 2019))

Balakrishnan, R., & Sahoo, R. K. (2006). Lossless compression for large scale cluster logs. In *Proceedings 20th IEEE International Parallel & Distributed Processing Symposium* (p. 7). doi: 10.1109/IPDPS.2006.1639692

Barik, T., DeLine, R., Drucker, S. M., & Fisher, D. (2016). The bones of the system: a case study of logging and telemetry at microsoft. In *Proceedings of the 38th International Conference on Software Engineering, ICSE 2016, may 14-22, 2016 - companion volume* (pp. 92–101). doi: 10.1145/2889160.2889231

Chen, A., Chen, T., & Wang, S. (2021). Pathidea: Improving information retrieval-based bug localization by re-constructing execution paths using logs. *IEEE Transactions on Software Engineering*, 1-1. doi: 10.1109/TSE.2021.3071473

Chen, A. R., Chen, T. P., & Wang, S. (2021). Demystifying the challenges and benefits of analyzing user-reported logs in bug reports. *Empirical Software Engineering*, *26*, 8. doi: 10.1007/s10664-020-09893-w

Chen, J., Shang, W., Hassan, A. E., Wang, Y., & Lin, J. (2019). An experience report of generating load tests using log-recovered workloads at varying granularities of user behaviour. In *Proceedings of the 34th IEEE/ACM International Conference on Automated Software Engineering* (pp. 669–681). doi: 10.1109/ASE.2019.00068

Chen, T.-H., Shang, W., Hassan, A. E., Nasser, M., & Flora, P. (2016). Cacheoptimizer: Helping developers configure caching frameworks for hibernate-based database-centric web applications. In *Proceedings of the 24th ACM SIGSOFT International Symposium on Foundations of Software Engineering* (pp. 666–677). doi: 10.1145/2950290.2950303

Chen, T.-H., Syer, M. D., Shang, W., Jiang, Z. M., Hassan, A. E., Nasser, M., & Flora, P. (2017). Analytics-driven load testing: An industrial experience report on load testing of large-scale systems. In *Proceedings of the 39th International Conference on Software Engineering: Software Engineering in Practice track* (pp. 243–252). doi: 10.1109/ICSE-SEIP.2017.26

Chow, M., Meisner, D., Flinn, J., Peek, D., & Wenisch, T. F. (2014). The mystery machine: End-to-end performance analysis of large-scale internet services. In *11th USENIX Symposium on Operating Systems Design and Implementation, OSDI '14, october 6-8, 2014*. (pp. 217–231). doi: 10.5555/2685048.2685066

Christensen, R., & Li, F. (2013). Adaptive log compression for massive log data. In *Proceedings of the 2013 ACM SIGMOD International Conference on Management of Data* (pp. 1283–1284). doi: 10.1145/2463676.2465341

Cito, J., Leitner, P., Fritz, T., & Gall, H. C. (2015). The making of cloud applications: an empirical study on software development for the cloud. In *Proceedings of the 2015 10th Joint Meeting on Foundations of Software Engineering, ESEC/FSE 2015, august 30 - september 4, 2015* (pp. 393–403). doi: 10.1145/2786805.2786826

Dai, H., Li, H., Shang, W., Chen, T.-H., & Chen, C.-S. (2020). Logram: Efficient log parsing using n-gram dictionaries. *IEEE Transactions on Software Engineering*, 1-1. doi: 10.1109/TSE.2020.3007554

Ding, R., Zhou, H., Lou, J., Zhang, H., Lin, Q., Fu, Q., ... Xie, T. (2015). Log2: A cost-aware logging mechanism for performance diagnosis. In *2015 USENIX Annual Technical Conference, USENIX ATC '15, july 8-10* (pp. 139–150). doi: 10.5555/2813767.2813778

Du, M., Li, F., Zheng, G., & Srikumar, V. (2017). 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). doi: 10.1145/3133956.3134015

Elastic. (n.d.). *Elastic*. https://www.elastic.co/. (Last accessed May 16, 2020.)

ElasticSearch. (n.d.). *Open-source log storage.* https://www.elastic.co/products/elasticsearch. (Last accessed May 16, 2020.)

Feng, B., Wu, C., & Li, J. (2016). MLC: an efficient multi-level log compression method for cloud backup systems. In *2016 IEEE TrustCom/BigDataSE/ISPA, august 23-26, 2016* (pp. 1358–1365). doi: 10.1109/TrustCom.2016.0215

Fu, Q., Lou, J.-G., Lin, Q., Ding, R., Zhang, D., & Xie, T. (2013). Contextual analysis of program logs for understanding system behaviors. In *Proceedings of the 10th Working Conference on Mining Software Repositories* (pp. 397–400). doi: 10.1109/MSR.2013.6624054

Fu, Q., Lou, J.-G., Wang, Y., & Li, J. (2009). Execution anomaly detection in distributed systems through unstructured log analysis. In *Proceedings of the 9th IEEE International Conference on Data Mining* (pp. 149–158). doi: 10.1109/ICDM.2009.60

Fu, Q., Zhu, J., Hu, W., Lou, J.-G., Ding, R., Lin, Q., ... Xie, T. (2014). Where do developers log? an empirical study on logging practices in industry. In *Companion Proceedings of the 36th International Conference on Software Engineering* (pp. 24–33). doi: 10.1145/2591062.2591175

Hassan, A. E., Martin, D. J., Flora, P., Mansfield, P., & Dietz, D. (2008). An industrial case study of customizing operational profiles using log compression. In *Proceedings of the 30th International Conference on Software Engineering* (pp. 713–723). doi: 10.1145/1368088.1379445

Hätönen, K., Boulicaut, J. F., Klemettinen, M., Miettinen, M., & Masson, C. (2003). Comprehensive log compression with frequent patterns. In *International Conference on Data Warehousing and Knowledge Discovery* (pp. 360–370). doi: 10.1007/978-3-540-45228-7_36

He, P., Chen, Z., He, S., & Lyu, M. R. (2018). Characterizing the natural language descriptions in software logging statements. In *Proceedings of the 33rd ACM/IEEE International Conference on Automated Software Engineering* (pp. 178–189). doi: 10.1145/3238147.3238193

He, P., Zhu, J., Zheng, Z., & Lyu, M. R. (2017). Drain: An online log parsing approach with fixed depth tree. In *2017 IEEE International Conference on Web Services (ICWS)* (pp. 33–40). doi: 10.1109/ICWS.2017.13

He, S., Lin, Q., Lou, J.-G., Zhang, H., Lyu, M. R., & Zhang, D. (2018). 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). doi: 10.1145/3236024.3236083

Jiang, Z. M., & Hassan, A. E. (2015). A survey on load testing of large-scale software systems. *IEEE Transactions on Software Engineering*, 1091–1118. doi: 10.1109/TSE.2015.2445340

Jiang, Z. M., Hassan, A. E., Hamann, G., & Flora, P. (2008a). An automated approach for abstracting execution logs to execution events. *Journal of Software Maintenance*, 249–267. doi: 10.5555/1400155.1400158

Jiang, Z. M., Hassan, A. E., Hamann, G., & Flora, P. (2008b). Automatic identification of load testing problems. In *Proceedings of the 2008 IEEE International Conference on Software Maintenance* (pp. 307–316). doi: 10.1109/ICSM.2008.4658079

LaToza, T. D., & Myers, B. A. (2010). Developers ask reachability questions. In *Proceedings of the 32nd ACM/IEEE International Conference on Software Engineering* (pp. 185–194). doi: 10.1145/1806799.1806829

Li, H., Shang, W., Adams, B., Sayagh, M., & Hassan, A. E. (2020). A qualitative study of the benefits and costs of logging from developers' perspectives. *IEEE Transactions on Software Engineering*, 1-1. doi: 10.1109/TSE.2020.2970422

Li, Y., Jiang, Z. M., Li, H., Hassan, A. E., He, C., Huang, R., … Chen, P. (2020). Predicting node failures in an ultra-large-scale cloud computing platform: an aiops solution. *ACM Transactions on Software Engineering and Methodology*. doi: 10.1145/3385187

Lin, Q., Zhang, H., Lou, J.-G., Zhang, Y., & Chen, X. (2016). Log clustering based problem identification for online service systems. In *Proceedings of the 38th International Conference on Software Engineering Companion* (p. 102–111). doi: 10.1145/2889160.2889232

Liu, J., Zhu, J., He, S., He, P., Zheng, Z., & Lyu, M. R. (2019). Logzip: Extracting hidden structures via iterative clustering for execution log compression. In *Proceedings of the 34th IEEE/ACM International Conference on Automated Software Engineering* (pp. 863–873). doi: 10.1109/ASE.2019.00085

Lou, J., Fu, Q., Yang, S., Xu, Y., & Li, J. (2010). Mining invariants from console logs for system problem detection. In *2010 USENIX Annual Technical Conference, june 23-25, 2010* (p. 24). doi: 10.5555/1855840.1855864

Mell, P., & Harang, R. E. (2014). Lightweight packing of log files for improved compression in mobile tactical networks. In *Military Communications Conference (MILCOM), 2014 IEEE* (pp. 192–197). doi: 10.1109/MILCOM.2014.37

Nagappan, M., Wu, K., & Vouk, M. A. (2009). Efficiently extracting operational profiles from execution logs using suffix arrays. In *Proceedings of the 20th IEEE International Conference on Software Reliability Engineering* (pp. 41–50). doi: 10.1109/ISSRE.2009.23

Nagaraj, K., Killian, C. E., & Neville, J. (2012). Structured comparative analysis of systems logs to diagnose performance problems. In *Proceedings of the 9th USENIX Symposium on Networked Systems Design and Implementation, NSDI 2012, april 25-27, 2012* (pp. 353–366).

Nageswaran, P. (1999, November 23). *Method, apparatus and computer program product for dynamically managing a thread pool of reusable threads in a computer system.* (US Patent 5,991,792)

Oliner, A., Ganapathi, A., & Xu, W. (2012). Advances and challenges in log analysis. *Communications of the ACM*, 55–61. doi: 10.1145/2076450.2076466

Oliner, A. J., & Stearley, J. (2007). What supercomputers say: A study of five system logs. In *The 37th Annual IEEE/IFIP International Conference on Dependable Systems and Networks, DSN 2007, 25-28 june 2007, proceedings* (pp. 575–584). doi: 10.1109/DSN.2007.103

Otten, F. J. (2008). *Using semantic knowledge to improve compression on log files* (Unpublished doctoral dissertation). Rhodes University.

Reiss, C., Wilkes, J., & Hellerstein, J. L. (2011). *Google cluster-usage traces: format + schema* (Technical Report). Mountain View, CA, USA: Google Inc. (Revised 2014-11-17 for version 2.1. Posted at https://github.com/google/cluster-data)

Schroeder, B., & Gibson, G. A. (2007). Disk failures in the real world: What does an MTTF of 1, 000, 000 hours mean to you? In *5th USENIX Conference on File and Storage Technologies, FAST 2007, february 13-16, 2007* (pp. 1–16). doi: 10.5555/1267903.1267904

Shang, W., Jiang, Z. M., Hemmati, H., Adams, B., Hassan, A. E., & Martin, P. (2013). Assisting developers of big data analytics applications when deploying on hadoop clouds. In *2013 35th International Conference on Software Engineering (ICSE)* (pp. 402–411). doi: 10.1109/ICSE.2013.6606586

Skibiński, P., & Swacha, J. (2007). Fast and efficient log file compression. In *CEUR Workshop Proceedings of the 11th East-European Conference on Advances in Databases and Information Systems* (pp. 330–342).

Splunk. (2017). *Turn machine data into answers.* https://www.splunk.com. (Last accessed May 16, 2020.)

Syer, M. D., Jiang, Z. M., Nagappan, M., Hassan, A. E., Nasser, M., & Flora, P. (2013). Leveraging performance counters and execution logs to diagnose memory-related performance issues. In *Proceedings of the 2013 IEEE International Conference on Software Maintenance* (pp. 110–119). doi: 10.1109/ICSM.2013.22

Syer, M. D., Jiang, Z. M., Nagappan, M., Hassan, A. E., Nasser, M., & Flora, P. (2014). Continuous validation of load test suites. In *Proceedings of the 5th ACM/SPEC International Conference on Performance Engineering* (pp. 259–270). doi: 10.1145/2568088.2568101

Tan, J., Pan, X., Kavulya, S., Gandhi, R., & Narasimhan, P. (2008). Salsa: analyzing logs as state machines. In *Proceedings of the 1st USENIX Conference on Analysis of System Logs* (pp. 6–6). doi: 10.5555/1855886.1855892

Virtanen, P., Gommers, R., Oliphant, T. E., Haberland, M., Reddy, T., Cournapeau, D., ... Contributors, S. . . (2020). SciPy 1.0: Fundamental Algorithms for Scientific Computing in Python. *Nature Methods*, *17*, 261–272. doi: https://doi.org/10.1038/s41592-019-0686-2

Xu, W., Huang, L., Fox, A., Patterson, D., & Jordan, M. I. (2009a). Detecting large-scale system problems by mining console logs. In *Proceedings of the ACM SIGOPS 22nd Symposium on Operating Systems Principles* (pp. 117–132). doi: 10.1145/1629575.1629587

Xu, W., Huang, L., Fox, A., Patterson, D. A., & Jordan, M. I. (2009b). Online system problem detection by mining patterns of console logs. In *ICDM 2009, Ninth IEEE International Conference on Data Mining, 6-9 december 2009* (pp. 588–597). doi: 10.1109/ICDM.2009.19

Yao, K., de Pádua, G. B., Shang, W., Sporea, C., Toma, A., & Sajedi, S. (2020). Log4perf: suggesting and updating logging locations for web-based systems' performance monitoring. *Empirical Software Engineering*, 488–531. doi: 10.1007/s10664-019-09748-z

Yao, K., Li, H., Shang, W., & Hassan, A. E. (2020). A study of the performance of general compressors on log files. *Empirical Software Engineering*, 1-1. doi: 10.1007/s10664-020-09822-x

Yuan, D., Mai, H., Xiong, W., Tan, L., Zhou, Y., & Pasupathy, S. (2010). Sherlog: Error diagnosis by connecting clues from run-time logs. In *Proceedings of the Fifteenth International Conference on Architectural Support for Programming Languages and Operating Systems* (pp. 143–154). doi: 10.1145/1736020.1736038

Zhu, J., He, S., Liu, J., He, P., Xie, Q., Zheng, Z., & Lyu, M. R. (2019). Tools and benchmarks for automated log parsing. In *Proceedings of the 41st International Conference on Software Engineering: Software Engineering in Practice* (pp. 121–130). doi: 10.1109/ICSE-SEIP.2019.00021