LogAssist: Assisting Log Analysis Through Log Summarization

Steven Locke, *Student Member, IEEE,* Heng Li, *Member, IEEE,* Tse-Hsun (Peter) Chen, *Member, IEEE,* Weiyi Shang, *Member, IEEE,* and Wei Liu, *Student Member, IEEE*

Abstract—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 paper, 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.

Index Terms—Log analysis, log compression, n-gram modeling, log abstraction, workflow characterization, log reduction

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 [1], [2], [3]. Therefore, logs are widely leveraged by practitioners in software development, operation, and maintenance tasks, such as failure diagnosis [4], [5], [6], anomaly detection [7], [8], [9], [10], [11], [12], performance analysis [13], [14], [15], [16], and system comprehension [4], [17].

Despite their importance, the enormous sizes (e.g., tens or hundreds of gigabytes) of logs [18], [19] have become a major obstacle for logs analysis [2], [1], [20], [21], [6]. 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 [22], [23]. 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 [6].
- **Rapidly growing log size.** Large-scale systems (e.g., cloud platforms) generate tens of gigabytes to terabytes of logs daily [24], [25], [20], 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 [26], [27]. 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 [28], [23]. 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 [29] and ELK [30] 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

- *S. Locke, T. Chen, and W. Liu are with Software PErformance, Analysis, and Reliability (SPEAR) lab, Concordia University, Montréal, Quebec, Canada.*
  W. Shang is with the Department of Computer Science and Software Engineering, Concordia University, Montréal, Quebec, Canada.
  E-mail: {s_loc, peterc, shang, w_liu201}@encs.concordia.ca
- *H. Li is with the Department of Computer Engineering and Software Engineering, Polytechnique, Montréal, Quebec, Canada*
  Email: heng.li@polymtl.ca

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):

RQ1 *How well can logs be compressed into re-occurring event sequences?* We quantitatively examine how effectively *LogAssist* can compress raw logs into concise representations.

RQ2 *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.

RQ3 *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.

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

2 Motivating Examples

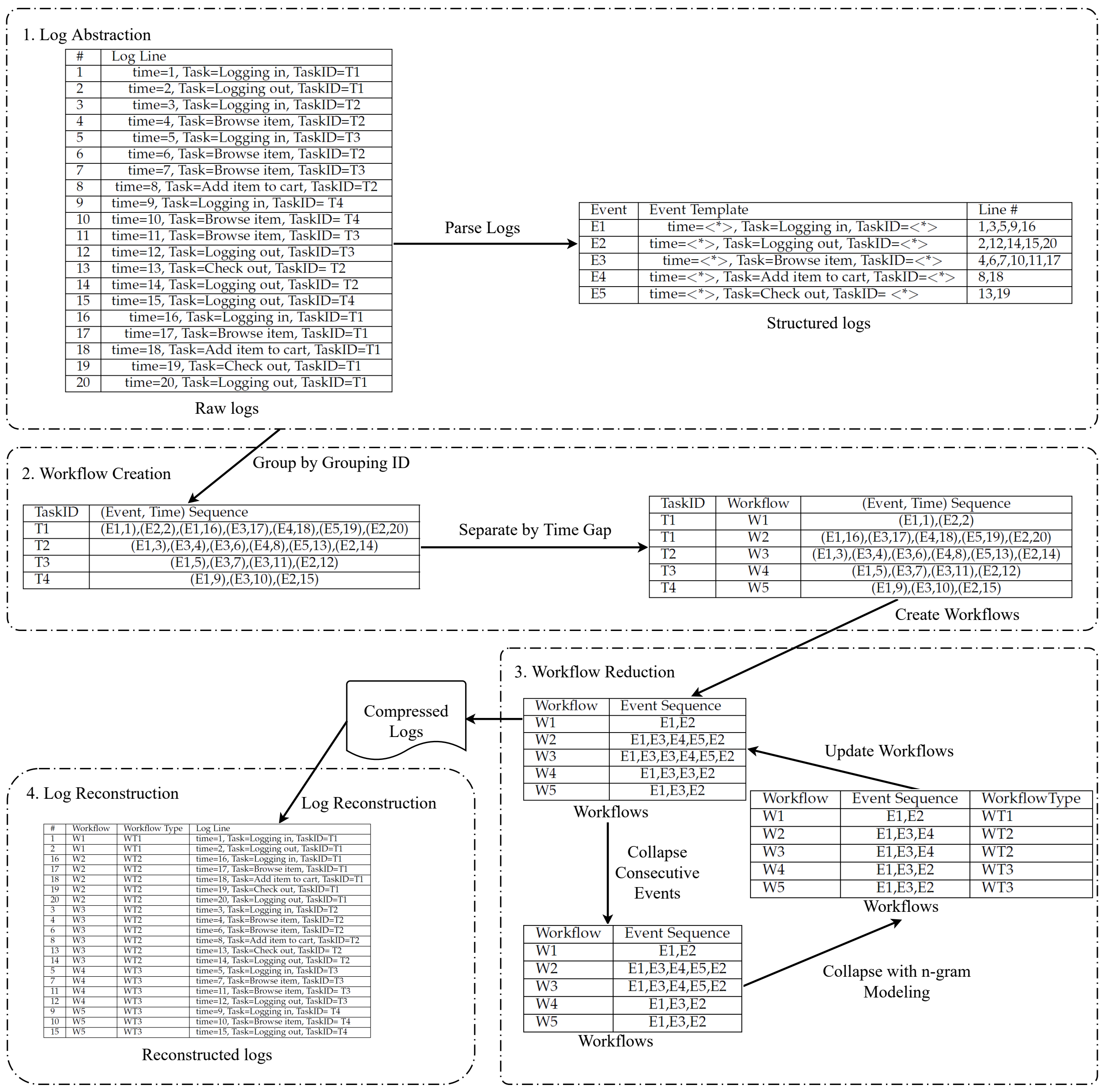
To illustrate the challenges that practitioners face during log analysis, we present motivating examples of using logs in three hypothetical, yet realistic situations on a large-scale enterprise system. The system is composed of several large components. Each component can be distributed in different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

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 [17], [31], [32]. 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 [33], [6], [34], [35]. 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.

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.

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 [6], [34]. 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.

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.



Raw logs

#	Log Line
1	time=1, Task=Logging in, TaskID=T1
2	time=2, Task=Logging out, TaskID=T1
3	time=3, Task=Logging in, TaskID=T2
4	time=4, Task=Browse item, TaskID=T2
5	time=5, Task=Logging in, TaskID=T3
6	time=6, Task=Browse item, TaskID=T2
7	time=7, Task=Browse item, TaskID=T3
8	time=8, Task=Add item to cart, TaskID=T2
9	time=9, Task=Logging in, TaskID= T4
10	time=10, Task=Browse item, TaskID= T4
11	time=11, Task=Browse item, TaskID= T3
12	time=12, Task=Logging out, TaskID=T3
13	time=13, Task=Check out, TaskID= T2
14	time=14, Task=Logging out, TaskID= T2
15	time=15, Task=Logging out, TaskID=T4
16	time=16, Task=Logging in, TaskID=T1
17	time=17, Task=Browse item, TaskID=T1
18	time=18, Task=Add item to cart, TaskID=T1
19	time=19, Task=Check out, TaskID=T1
20	time=20, Task=Logging out, TaskID=T1

Structured logs

Event	Event Template	Line #
E1	time=<*>, Task=Logging in, TaskID=<*>	1,3,5,9,16
E2	time=<*>, Task=Logging out, TaskID=<*>	2,12,14,15,20
E3	time=<*>, Task=Browse item, TaskID=<*>	4,6,7,10,11,17
E4	time=<*>, Task=Add item to cart, TaskID=<*>	8,18
E5	time=<*>, Task=Check out, TaskID= <*>	13,19

TaskID	(Event, Time) Sequence
T1	(E1,1),(E2,2),(E1,16),(E3,17),(E4,18),(E5,19),(E2,20)
T2	(E1,3),(E3,4),(E3,6),(E4,8),(E5,13),(E2,14)
T3	(E1,5),(E3,7),(E3,11),(E2,12)
T4	(E1,9),(E3,10),(E2,15)

TaskID	Workflow	(Event, Time) Sequence
T1	W1	(E1,1),(E2,2)
T1	W2	(E1,16),(E3,17),(E4,18),(E5,19),(E2,20)
T2	W3	(E1,3),(E3,4),(E3,6),(E4,8),(E5,13),(E2,14)
T3	W4	(E1,5),(E3,7),(E3,11),(E2,12)
T4	W5	(E1,9),(E3,10),(E2,15)

Workflows

Workflow	Event Sequence
W1	E1,E2
W2	E1,E3,E4,E5,E2
W3	E1,E3,E3,E4,E5,E2
W4	E1,E3,E3,E2
W5	E1,E3,E2

Workflows

Workflow	Event Sequence
W1	E1,E2
W2	E1,E3,E4,E5,E2
W3	E1,E3,E4,E5,E2
W4	E1,E3,E2
W5	E1,E3,E2

Workflows

Workflow	Event Sequence	WorkflowType
W1	E1,E2	WT1
W2	E1,E3,E4	WT2
W3	E1,E3,E4	WT2
W4	E1,E3,E2	WT3
W5	E1,E3,E2	WT3

Reconstructed logs

#	Workflow	Workflow Type	Log Line
1	W1	WT1	time=1, Task=Logging in, TaskID=T1
2	W1	WT1	time=2, Task=Logging out, TaskID=T1
16	W2	WT2	time=16, Task=Logging in, TaskID=T1
17	W2	WT2	time=17, Task=Browse item, TaskID=T1
18	W2	WT2	time=18, Task=Add item to cart, TaskID=T1
19	W2	WT2	time=19, Task=Check out, TaskID=T1
20	W2	WT2	time=20, Task=Logging out, TaskID=T1
3	W3	WT2	time=3, Task=Logging in, TaskID=T2
4	W3	WT2	time=4, Task=Browse item, TaskID=T2
6	W3	WT2	time=6, Task=Browse item, TaskID=T2
8	W3	WT2	time=8, Task=Add item to cart, TaskID=T2
13	W3	WT2	time=13, Task=Check out, TaskID= T2
14	W3	WT2	time=14, Task=Logging out, TaskID= T2
5	W4	WT3	time=5, Task=Logging in, TaskID=T3
7	W4	WT3	time=7, Task=Browse item, TaskID=T3
11	W4	WT3	time=11, Task=Browse item, TaskID= T3
12	W4	WT3	time=12, Task=Logging out, TaskID=T3
9	W5	WT3	time=9, Task=Logging in, TaskID= T4
10	W5	WT3	time=10, Task=Browse item, TaskID= T4
15	W5	WT3	time=15, Task=Logging out, TaskID=T4

Fig. 1. The overall flow of our approach *LogAssist* with a running example demonstrating its steps.

3 THE DESIGN OF *LogAssist*

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

Figure 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 [7], [31], [36]. Log abstraction is widely used to categorize raw log lines [17], [31], [37], [38], [39] 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 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* [26], 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 one of the state-of-the-art approaches for log abstraction [36]. 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 [26]. As demonstrated in Figure 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 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 [10], [11], [31], [32], 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 [6], [37], [40], [31]. Hence, in this step, *LogAssist* creates workflows from the parsed log events.

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 [11], [10], [31], [32]. An example of intermixing events can be seen in Figure 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 1), the grouping ID is “TaskID”.

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 repeated by different occurrences of the same type of workflow or be reused by different types of workflows (e.g., each thread in a thread pool might be reused, so the same thread ID will appear multiple times) [41]. 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 reusing the grouping ID will lead to larger time differences. We use a *find_peaks* algorithm from the signal processing domain [42] 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 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 [10], [11], [4]. Such repetitive log events may mask real problems in the logs or introduce additional challenges in log analysis [7], [17], [43], [44]. 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.

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 [17]. In the running example (Figure 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*. Both instances of 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.

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 in [17], we apply n-gram modeling to apply further reduction 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 event allows *LogAssist* to reduce all instances of these workflow types to the same common workflow type format 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, $*$ 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 [22] 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).

***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.

3.5 An Exemplar Usage Scenario of *LogAssist*

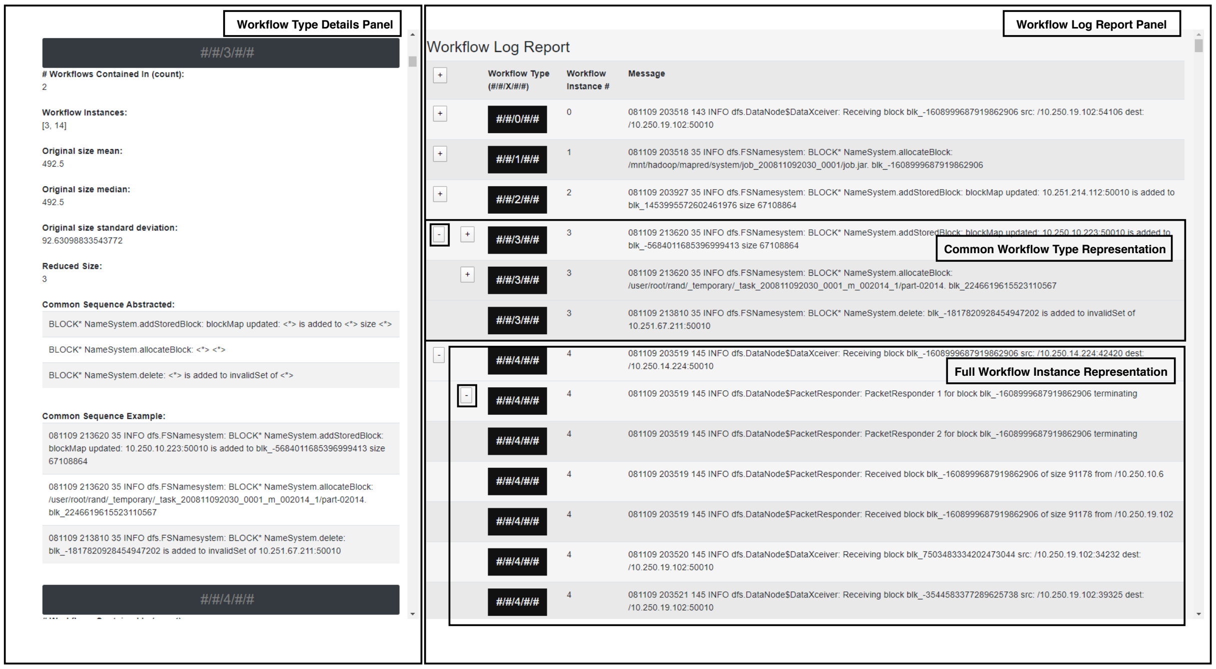
We implemented a web-based graphical user interface as shown in Figure 2. The *Workflow Type Details Panel* to the left shows the statistics of a unique workflow type (e.g., the number of workflows that belong to this unique workflow type, the number of events in the unique workflow type, the size of 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 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 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 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.

4 Evaluation

In this section, 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 [36], 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 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


Workflow Type Details Panel
Workflow Log Report Panel
Common Workflow Type Representation
Full Workflow Instance Representation

#/#/3/#/#

# Workflows Contained In (count):
2

Workflow Instances:
[3, 14]

Original size mean:
492.5

Original size median:
492.5

Original size standard deviation:
92.63098833543772

Reduced Size:
3

Common Sequence Abstracted:

```
BLOCK* NameSystem.addStoredBlock: blockMap updated: <*> is added to <*> size <*>
BLOCK* NameSystem.allocateBlock: <*> <*>
BLOCK* NameSystem.delete: <*> is added to invalidSet of <*>
```

Common Sequence Example:

```
081109 213620 35 INFO dfs.FSNamesystem: BLOCK* NameSystem.addStoredBlock: blockMap updated: 10.250.10.223:50010 is added to blk_-5684011685396999413 size 67108864
081109 213620 35 INFO dfs.FSNamesystem: BLOCK* NameSystem.allocateBlock: /user/root/rand/_temporary/_task_200811092030_0001_m_002014_1/part-02014. blk_2246619615523110567
081109 213810 35 INFO dfs.FSNamesystem: BLOCK* NameSystem.delete: blk_-1817820928454947202 is added to invalidSet of 10.251.67.211:50010
```

#/#/4/#/#

Workflow Log Report

Workflow Type (#/#/X/#/#)	Workflow Instance #	Message
#/#/0/#/#	0	081109 203518 143 INFO dfs.DataNode$DataXceiver: Receiving block blk_-1608999687919862906 src: /10.250.19.102:54106 dest: /10.250.19.102:50010
#/#/1/#/#	1	081109 203518 35 INFO dfs.FSNamesystem: BLOCK* NameSystem.allocateBlock: /mnt/hadoop/mapred/system/job_200811092030_0001/job.jar. blk_-1608999687919862906
#/#/2/#/#	2	081109 203927 35 INFO dfs.FSNamesystem: BLOCK* NameSystem.addStoredBlock: blockMap updated: 10.251.214.112:50010 is added to blk_1453995572602461976 size 67108864
#/#/3/#/#	3	081109 213620 35 INFO dfs.FSNamesystem: BLOCK* NameSystem.addStoredBlock: blockMap updated: 10.250.10.223:50010 is added to blk_-5684011685396999413 size 67108864
#/#/3/#/#	3	081109 213620 35 INFO dfs.FSNamesystem: BLOCK* NameSystem.allocateBlock: /user/root/rand/_temporary/_task_200811092030_0001_m_002014_1/part-02014. blk_2246619615523110567
#/#/3/#/#	3	081109 213810 35 INFO dfs.FSNamesystem: BLOCK* NameSystem.delete: blk_-1817820928454947202 is added to invalidSet of 10.251.67.211:50010
#/#/4/#/#	4	081109 203519 145 INFO dfs.DataNode$DataXceiver: Receiving block blk_-1608999687919862906 src: /10.250.14.224:42420 dest: /10.250.14.224:50010
#/#/4/#/#	4	081109 203519 145 INFO dfs.DataNode$PacketResponder: PacketResponder 1 for block blk_-1608999687919862906 terminating
#/#/4/#/#	4	081109 203519 145 INFO dfs.DataNode$PacketResponder: PacketResponder 2 for block blk_-1608999687919862906 terminating
#/#/4/#/#	4	081109 203519 145 INFO dfs.DataNode$PacketResponder: Received block blk_-1608999687919862906 of size 91178 from /10.250.10.6
#/#/4/#/#	4	081109 203519 145 INFO dfs.DataNode$PacketResponder: Received block blk_-1608999687919862906 of size 91178 from /10.250.19.102
#/#/4/#/#	4	081109 203520 145 INFO dfs.DataNode$DataXceiver: Receiving block blk_7503483334202473044 src: /10.250.19.102:34232 dest: /10.250.19.102:50010
#/#/4/#/#	4	081109 203521 145 INFO dfs.DataNode$DataXceiver: Receiving block blk_-3544583377289625738 src: /10.250.19.102:39325 dest: /10.250.19.102:50010

Fig. 2. An exemplar web-based user interface of *LogAssist*.

TABLE 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

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.

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 [7], [10], [11]. 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 [2], [3]. 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. [17] 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. [17] also applied workflow reduction. Therefore, we use [17] as a baseline and compare it with *LogAssist*. Both *LogAssist* and [17] 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 Section 3.2. Additionally, while *LogAssist* and [17] both

TABLE 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%

TABLE 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%)

summarize event sequences (workflows) by collapsing consecutive repeating events, [17] applies 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.

[17] groups 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 [17] with *LogAssist*, we consider both with and without permutations for the two approaches, and report the reduction in unique workflow types and total log lines.

Evaluating the effect of n-gram modeling. Prior work [17] 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 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 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 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 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 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 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 sig-

TABLE 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

TABLE 5
A comparison between *LogAssist* and current state-of-the-art approach by Shang et al. [17] 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

nificantly 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 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 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 5 shows that both *LogAssist* and its variation with permutation grouping outperform [17]. As previously stated, due to differing goals between *LogAssist* and [17], 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 5 shows the comparison results. The findings indicate that in all cases, *LogAssist* outperforms [17] 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 workflow types for HDFS and Zookeeper when using *LogAssist*. Finally, comparing [17] 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 [17]. The reason is that [17] only reduces 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.

LogAssist is able to reduce the log events that practitioners need to investigate by 75.2% to 93.9%. Moreover, by using *LogAssist*, we can significantly reduce the unique workflow types by 70.2% to 89.8% in the open source systems. Our results indicate that there is a significant number of re-occurring log events in the studied systems, and reducing such information may further help practitioners with log analysis.

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 [17], [31], [32]. 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 [33], [40], [45]. 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 *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 [17] 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 [46], [47], [29]. 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

TABLE 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

TABLE 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 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%

event for each of the three categories, since those events are the ones that practitioners may need to spend the most time examining [17]. We list and explain the keywords that we use to search for log lines in each of the studied systems in Table 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) [33], [6], [40]. 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 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%.

LogAssist **dramatically compresses the searched-line-related workflows that need to be examined by practitioners (i.e., by up to 99% reduction).** Table 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 [6], [40], [33]). 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 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 percent-

TABLE 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%)

age of workflows 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 7 and Table 8, and the number of workflows and workflow types that contain these keywords.

LogAssist reduces the number of log lines that need to be examined by practitioners by up to 99%. Our results also indicate that there may be many different workflows that lead to a system runtime issue, and *LogAssist* can help practitioners identify such unique workflows.

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.

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, 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 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

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

TABLE 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

time when using *LogAssist*. However, the result shows a statistically significant reduction in the overall completion time of the tasks when using *LogAssist* ($p<0.01$).

LogAssist **improves the users' experience of performing the log analysis tasks.** As shown in Figure 3, 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 and comments 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.

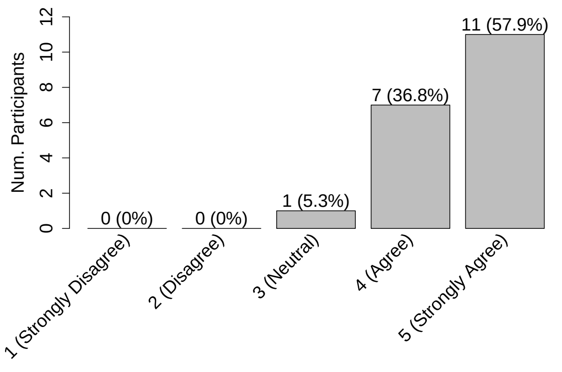


Fig. 3. User provided rating for the usefulness of *LogAssist*.

LogAssist provides, on average, a 42% improvement in log analysis speed when compared to performing the same analysis on raw logs alone. Participants rated *LogAssist* an average of 4.53 out of 5 (95% of the participants rate *LogAssist* as *agree* or *strongly agree*) for improving their experience of performing log analysis.

5 Lessons Learned

Logs are very repetitive, while most of the log information can be compressed without impacting the usefulness of logs. Prior research [48], [28], [23] 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 [17], [11], [7]. Prior work uses n-gram models to measure the repetitiveness of log data [23] or to identify the static parts of a log line [49]. 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 and strengthens 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 pay attention to assisting 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.

6 Threats to Validity

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

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 [36], our results may not be generalized to other systems. Future studies are needed to verify the effectiveness of our approach on other systems.

Construct validity. We evaluate our approach by following a prior study [17]. 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 paper 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 (Section 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.

7 Related Work

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

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 [7], [38], [39], [11], [31], [37]. Such anomalies may be an indication of the problem that happened during system execution. For example, Xu et al. [7] propose an approach to first group log lines using grouping ID and then apply principal component analysis to detect anomalies. Jiang et al. [11] groups log lines using grouping ID and apply z-stat to detect anomalies. Syer et al. [38], [39] use hierarchical clustering to identify anomalies in execution logs. Du et al. [37] leverage deep learning models (i.e., LSTM) to detect anomalies in log sequences. Chen et al. [31] 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.

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. [6] analyze log lines to uncover system execution paths in the source code. Tan et al. [40] analyze log lines by using state machines to model system execution. Chen et al. [50] leverage log lines to analyze system workflows and recommend where to place caches. Chen et al. [43] propose approaches to extract representative workflows from production logs to assist with load test design at different levels of granularity. Lin et al. [44] 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 [30] 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. [17]. While [17] seeks 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 Section 4.2) and their provided benefits. As discussed in Section 4.2, *LogAssist* provides a more concise form of logs than [17]. 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).

Log compression. Prior work [51], [52], [53], [54], [55], [56], [57], [28] 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 [54], [55] and log transposing [56]. Prior work also compresses logs by replacing long and repetitive text in log files with shorter representations [53], [51], [52], [57], [28]. For example, Otten et al. [51] transform 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. [28] 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 et al. [23] evaluates 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.

8 Conclusion

In this paper, 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 paper makes the following contributions:

- We propose a novel approach that effectively compresses raw logs into a concise form that simplifies practitioners' log analysis tasks.
- We demonstrate the importance of untangling raw logs into meaningful event sequences (i.e, workflows) and use statistical techniques (e.g., n-gram models) to identify re-occurring patterns of event sequences.
- We share the lessons that we learned from developing and adopting our approach, which can provide insights for researchers and practitioners who wish to develop similar tools to assist log analysis tasks.

References

[1] T. Barik, R. DeLine, S. M. Drucker, and D. Fisher, "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*, L. K. Dillon, W. Visser, and L. Williams, Eds., 2016, pp. 92–101.

[2] H. Li, W. Shang, B. Adams, M. Sayagh, and A. E. Hassan, "A qualitative study of the benefits and costs of logging from developers' perspectives," *IEEE Transactions on Software Engineering*, pp. 1–1, 2020.

[3] Q. Fu, J. Zhu, W. Hu, J.-G. Lou, R. Ding, Q. Lin, D. Zhang, and T. Xie, "Where do developers log? an empirical study on logging practices in industry," in *Companion Proceedings of the 36th International Conference on Software Engineering*, 2014, pp. 24–33.

[4] Q. Fu, J.-G. Lou, Q. Lin, R. Ding, D. Zhang, and T. Xie, "Contextual analysis of program logs for understanding system behaviors," in *Proceedings of the 10th Working Conference on Mining Software Repositories*, ser. MSR '13, 2013, pp. 397–400.

[5] "Automated root cause analysis for spark application failures - o'reilly media," (Accessed on 08/13/2019).

[6] D. Yuan, H. Mai, W. Xiong, L. Tan, Y. Zhou, and S. Pasupathy, "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*, ser. ASPLOS 15th, 2010, pp. 143–154.

[7] W. Xu, L. Huang, A. Fox, D. Patterson, and M. I. Jordan, "Detecting large-scale system problems by mining console logs," in *Proceedings of the ACM SIGOPS 22Nd Symposium on Operating Systems Principles*, ser. SOSP '09, 2009, pp. 117–132.

[8] W. Xu, L. Huang, A. Fox, D. A. Patterson, and M. I. Jordan, "Online system problem detection by mining patterns of console logs," in *ICDM 2009, The Ninth IEEE International Conference on Data Mining, 6-9 December 2009*, 2009, pp. 588–597.

[9] J. Lou, Q. Fu, S. Yang, Y. Xu, and J. Li, "Mining invariants from console logs for system problem detection," in *2010 USENIX Annual Technical Conference, June 23-25, 2010*, 2010.

[10] Q. Fu, J.-G. Lou, Y. Wang, and J. Li, "Execution anomaly detection in distributed systems through unstructured log analysis," in *Proceedings of the 9th IEEE International Conference on Data Mining*, ser. ICDM '09, 2009, pp. 149–158.

[11] Z. M. Jiang, A. E. Hassan, G. Hamann, and P. Flora, "Automatic identification of load testing problems," in *Proceedings of the 2008 IEEE International Conference on Software Maintenance*, ser. ICSM '08, 2008, pp. 307–316.

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

[13] K. Nagaraj, C. E. Killian, and J. Neville, "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*, 2012, pp. 353–366.

[14] M. Chow, D. Meisner, J. Flinn, D. Peek, and T. F. Wenisch, "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.*, 2014, pp. 217–231.

[15] K. Yao, G. B. de Pádua, W. Shang, C. Sporea, A. Toma, and S. Sajedi, "Log4perf: suggesting and updating logging locations for web-based systems' performance monitoring," *Empirical Software Engineering*, vol. 25, no. 1, pp. 488–531, 2020.

[16] R. Ding, H. Zhou, J. Lou, H. Zhang, Q. Lin, Q. Fu, D. Zhang, and T. Xie, "Log2: A cost-aware logging mechanism for performance diagnosis," in *2015 USENIX Annual Technical Conference, USENIX ATC '15, July 8-10*, S. Lu and E. Riedel, Eds., 2015, pp. 139–150.

[17] W. Shang, Z. M. Jiang, H. Hemmati, B. Adams, A. E. Hassan, and P. Martin, "Assisting developers of big data analytics applications when deploying on hadoop clouds," in *2013 35th International Conference on Software Engineering (ICSE)*, 2013, pp. 402–411.

[18] A. J. Oliner and J. Stearley, "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*, 2007, pp. 575–584.

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

[20] J. Cito, P. Leitner, T. Fritz, and H. C. Gall, "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*, E. D. Nitto, M. Harman, and P. Heymans, Eds., 2015, pp. 393–403.

[21] A. Oliner, A. Ganapathi, and W. Xu, "Advances and challenges in log analysis," *Commun. ACM*, vol. 55, no. 2, pp. 55–61, Feb. 2012.

[22] P. He, Z. Chen, S. He, and M. R. Lyu, "Characterizing the natural language descriptions in software logging statements," in *Proceedings of the 33rd ACM/IEEE International Conference on Automated Software Engineering*, ser. ASE 2018, 2018, pp. 178–189.

[23] K. Yao, H. Li, W. Shang, and A. E. Hassan, "A study of the performance of general compressors on log files," *Empirical Software Engineering*, pp. 1–1, 2020.

[24] Y. Li, Z. M. Jiang, H. Li, A. E. Hassan, C. He, R. Huang, Z. Zeng, M. Wang, and P. Chen, "Predicting node failures in an ultra-large-scale cloud computing platform: an aiops solution," *ACM Transactions on Software Engineering and Methodology*, 2020.

[25] C. Reiss, J. Wilkes, and J. L. Hellerstein, "Google cluster-usage traces: format + schema," Google Inc., Technical Report, Nov. 2011, revised 2014-11-17 for version 2.1. Posted at https://github.com/google/cluster-data.

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

[27] Z. M. Jiang, A. E. Hassan, G. Hamann, and P. Flora, "An automated approach for abstracting execution logs to execution events," *Journal of Software Maintenance*, vol. 20, no. 4, pp. 249–267, 2008.

[28] J. Liu, J. Zhu, S. He, P. He, Z. Zheng, and M. R. Lyu, "Logzip: Extracting hidden structures via iterative clustering for execution log compression," in *Proceedings of the 34th IEEE/ACM International Conference on Automated Software Engineering*. ACM, 2019.

[29] Splunk, "Turn machine data into answers," https://www.splunk.com, last accessed May 16 2020.

[30] Elastic, "Elastic," https://www.elastic.co/, last accessed May 16 2020.

[31] T.-H. Chen, M. D. Syer, W. Shang, Z. M. Jiang, A. E. Hassan, M. Nasser, and P. Flora, "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*, ser. ICSE-SEIP '17, 2017, pp. 243–252.

[32] Z. M. Jiang and A. E. Hassan, "A survey on load testing of large-scale software systems," *IEEE Transactions on Software Engineering*, vol. 41, no. 11, pp. 1091–1118, 2015.

[33] T. D. LaToza and B. A. Myers, "Developers ask reachability questions," in *Proceedings of the 32Nd ACM/IEEE International Conference on Software Engineering*, ser. ICSE '10, 2010, pp. 185–194.

[34] A. R. Chen, T. P. Chen, and S. Wang, "Demystifying the challenges and benefits of analyzing user-reported logs in bug reports," *Empirical Software Engineering*, vol. 26, no. 1, p. 8, 2021.

[35] A. Chen, T. Chen, and S. Wang, "Pathidea: Improving information retrieval-based bug localization by re-constructing execution paths using logs," *IEEE Transactions on Software Engineering*, no. 01, pp. 1–1, 2021.

[36] J. Zhu, S. He, J. Liu, P. He, Q. Xie, Z. Zheng, and M. R. Lyu, "Tools and benchmarks for automated log parsing," in *Proceedings of the 41st International Conference on Software Engineering: Software Engineering in Practice*, ser. ICSE-SEIP '19, 2019, pp. 121–130.

[37] M. Du, F. Li, G. Zheng, and V. Srikumar, "Deeplog: Anomaly detection and diagnosis from system logs through deep learning," in *Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security*, ser. CCS '17, 2017, p. 1285–1298.

[38] M. D. Syer, Z. M. Jiang, M. Nagappan, A. E. Hassan, M. Nasser, and P. Flora, "Leveraging performance counters and execution logs to diagnose memory-related performance issues," in *Proceedings of the 2013 IEEE International Conference on Software Maintenance*, ser. ICSM '13, 2013, pp. 110–119.

[39] ——, "Continuous validation of load test suites," in *Proceedings of the 5th ACM/SPEC International Conference on Performance Engineering*, ser. ICPE '14, 2014, pp. 259–270.

[40] J. Tan, X. Pan, S. Kavulya, R. Gandhi, and P. Narasimhan, "Salsa: analyzing logs as state machines," in *Proceedings of the 1st USENIX conference on Analysis of system logs*, ser. WASL'08, 2008, pp. 6–6.

[41] P. Nageswaran, "Method, apparatus and computer program product for dynamically managing a thread pool of reusable threads in a computer system," Nov. 23 1999, uS Patent 5,991,792.

[42] P. Virtanen, R. Gommers, T. E. Oliphant, M. Haberland, T. Reddy, D. Cournapeau, E. Burovski, P. Peterson, W. Weckesser, J. Bright, S. J. van der Walt, M. Brett, J. Wilson, K. Jarrod Millman, N. Mayorov, A. R. J. Nelson, E. Jones, R. Kern, E. Larson, C. Carey, İ. Polat, Y. Feng, E. W. Moore, J. Vand erPlas, D. Laxalde, J. Perktold, R. Cimrman, I. Henriksen, E. A. Quintero, C. R. Harris, A. M. Archibald, A. H. Ribeiro, F. Pedregosa, P. van Mulbregt, and S. . . Contributors, "SciPy 1.0: Fundamental Algorithms for Scientific Computing in Python," *Nature Methods*, vol. 17, pp. 261–272, 2020.

[43] J. Chen, W. Shang, A. E. Hassan, Y. Wang, and J. Lin, "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*, ser. ASE '19, 2019, p. 669–681.

[44] Q. Lin, H. Zhang, J.-G. Lou, Y. Zhang, and X. Chen, "Log clustering based problem identification for online service systems," in *Proceedings of the 38th International Conference on Software Engineering Companion*, ser. ICSE '16, 2016, p. 102–111.

[45] M. Nagappan, K. Wu, and M. A. Vouk, "Efficiently extracting operational profiles from execution logs using suffix arrays," in *Proceedings of the 20th IEEE International Conference on Software Reliability Engineering*, ser. ISSRE'09, 2009, pp. 41–50.

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

[47] ElasticSearch, "Open-source log storage," https://www.elastic.co/products/elasticsearch, last accessed May 16 2020.

[48] A. E. Hassan, D. J. Martin, P. Flora, P. Mansfield, and D. Dietz, "An industrial case study of customizing operational profiles using log compression," in *Proceedings of the 30th International Conference on Software Engineering*, ser. ICSE '08, 2008, p. 713–723.

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

[50] T.-H. Chen, W. Shang, A. E. Hassan, M. Nasser, and P. Flora, "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*, ser. FSE 2016, 2016, p. 666–677.

[51] F. J. Otten, “Using semantic knowledge to improve compression on log files,” Ph.D. dissertation, Rhodes University, 2008.

[52] P. Skibiński and J. Swacha, “Fast and efficient log file compression,” in *CEUR Workshop Proceedings of the 11th East-European Conference on Advances in Databases and Information Systems*, ser. ADBIS’07. ACM, 2007, pp. 330–342.

[53] R. Balakrishnan and R. K. Sahoo, “Lossless compression for large scale cluster logs,” in *Proceedings 20th IEEE International Parallel & Distributed Processing Symposium*. IEEE, 2006, p. 435.

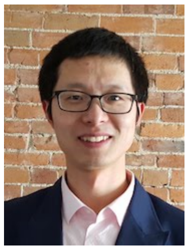
[54] R. Christensen and F. Li, “Adaptive log compression for massive log data.” in *SIGMOD Conference*. ACM, 2013, pp. 1283–1284.

[55] B. Feng, C. Wu, and J. Li, “MLC: an efficient multi-level log compression method for cloud backup systems,” in *2016 IEEE Trustcom/BigDataSE/ISPA, August 23-26, 2016*, 2016, pp. 1358–1365.

[56] P. Mell and R. E. Harang, “Lightweight packing of log files for improved compression in mobile tactical networks,” in *Military Communications Conference (MILCOM), 2014 IEEE*. IEEE, 2014, pp. 192–197.

[57] K. Hätönen, J. F. Boulicault, M. Klemettinen, M. Miettinen, and C. Masson, “Comprehensive log compression with frequent patterns,” in *International Conference on Data Warehousing and Knowledge Discovery*. Springer, 2003, pp. 360–370.

Steven Locke Steven Locke is a Master’s student in the Department of Computer Science and Software Engineering at Concordia University, Montreal, Canada, supervised by Dr. Tse-Hsun (Peter) Chen. His research lies within Software Engineering, with special interests in software log mining, mining software repositories, and the application of AI in software engineering. He obtained his Bachelor of Engineering in Software Engineering from Concordia University. Contact him at: s_loc@encs.concordia.ca



Heng Li is an Assistant Professor in the Department of Computer Engineering and Software Engineering at Polytechnique Montreal, Montreal, Canada, where he leads the Maintenance, Operations and Observation of Software with intelligencE (MOOSE) lab. He obtained his Ph.D. from the School of Computing, Queen’s University (Canada), M.Sc. from Fudan University (China), and B.Eng. from Sun Yat-sen University (China). He also worked at Synopsys as a software engineer for two years and worked at BlackBerry as a software performance engineer for another two years. His research interests lie within Software Engineering, in particular, software observability, intelligent operations of software systems, software log mining, software performance engineering, and mining software repositories. Contact him at: heng.li@polymtl.ca; https://www.hengli.org.

Tse-Hsun (Peter) Chen Tse-Hsun (Peter) Chen is an Assistant Professor in the Department of Computer Science and Software Engineering at Concordia University, Montreal, Canada. He leads the Software PErformance, Analysis, and Reliability (SPEAR) Lab, which focuses on conducting research on performance engineering, program analysis, log analysis, production debugging, and mining software repositories. His work has been published in flagship conferences and journals such as ICSE, FSE, TSE, EMSE, and MSR. He serves regularly as a program committee member of international conferences in the field of software engineering, such as ASE, ICSE, ICSME, SANER, and ICPC, and he is a regular reviewer for software engineering journals such as EMSE and TSE. Dr. Chen obtained his BSc from the University of British Columbia, and MSc and PhD from Queen’s University. Besides his academic career, Dr. Chen also worked as a software performance engineer at BlackBerry for over four years. Early tools developed by Dr. Chen were integrated into industrial practice for ensuring the quality of large-scale enterprise systems. More information at: http://petertsehsun.github.io/.

Weiyi Shang Weiyi Shang is an Associate Professor and Concordia University Research Chair in Ultra-large-scale Systems at the Department of Computer Science and Software Engineering at Concordia University, Montreal. He has received his Ph.D. and M.Sc. degrees from Queens University (Canada) and he obtained B.Eng. from Harbin Institute of Technology. His research interests include big data software engineering, software engineering for ultra-largescale systems, software log mining, empirical software engineering, and software performance engineering. His work has been published at premier venues such as ICSE, FSE, ASE, ICSME, MSR and WCRE, as well as in major journals such as TSE, EMSE, JSS, JSEP and SCP. His work has won premium awards, such as SIGSOFT Distinguished paper award at ICSE 2013 and best paper award at WCRE 2011. His industrial experience includes helping improve the quality and performance of ultra-large-scale systems in BlackBerry. Early tools and techniques developed by him are already integrated into products used by millions of users worldwide. Contact him at shang@encs.concordia.ca.

Wei Liu Wei Liu received the bachelor’s degree in computer science and the master’s degree in software engineering from Wuhan University of Technology, China, in 2012 and 2014, respectively. He is currently a Ph.D student at the Department of Computer Science and Software Engineering at Concordia University, Montreal, Canada. His research interests include software analysis, testing and performance. Prior to it, he worked at Alibaba as a software engineer for two years.