Noname manuscript No.
(will be inserted by the editor)

A Study of the Performance of General Compressors on Log Files

Kundi Yao · Heng Li · Weiyi Shang · Ahmed E. Hassan

Received: date / Accepted: date

Abstract Large-scale software systems and cloud services continue to produce a large amount of log data. Such log data is usually preserved for a long time (e.g., for auditing purposes). General compressors, like the *LZ77* compressor used in *gzip*, are usually used in practice to compress log data to reduce the cost of long-term storage. However, such general compressors do not consider the unique nature of log data. In this paper, we study the performance of general compressors on compressing log data relative to their performance on compressing natural language data. We used 12 widely used general compressors to compress nine log files that are collected based on surveying prior literature on text compression, log compression and log analysis. We observe that log data is more repetitive than natural language data, and that log data can be compressed and decompressed faster with higher compression ratios. Besides, the compressor with the highest compression ratio for natural language data is rarely the one for log data. Nevertheless, the compressors with the highest compression ratio for log data are rarely adopted in practice by current logging libraries and log management tools. We also observe that the peak compression and decompression speeds of general compressors on log data is often achieved with a small data size, while such size may not be used by log management tools. Finally, we observe that the optimal compression performance (measured by a combined compression performance score) of log data usually requires the compression level to be configured higher than the

Kundi Yao · Heng Li · Ahmed E. Hassan
Software Analysis and Intelligence Lab (SAIL)
Queen's University
Kingston, Ontario, Canada
E-mail: {kundi, hengli, ahmed}@cs.queensu.ca

Weiyi Shang
Department of Computer Science and Software Engineering
Concordia University
Montreal, Quebec, Canada
E-mail: shang@encs.concordia.ca

default level. Our findings call for careful consideration of choosing general compressors and their associated compression levels for log data in practice. In addition, our findings shed lights on the opportunities for future research on compressors that better suit the characteristics of log data.

Keywords Log compression · Software logging · Log management · Language model

1 Introduction

Log data is generated by logging statements that developers place into the source code for tracing, debugging and failure diagnosis [22, 34, 47, 49, 64, 71, 72]. Log data is usually the only source of information that enables practitioners to understand the field runtime behavior of a system [16, 39, 75, 77]. Besides, log data and their long-term archival are usually required for legal compliance [33]. As a result, large-scale software systems usually produce large volumes of log data every day. Such a large volume of data poses challenges for software practitioners to manage, analyze, and preserve such log data.

In recent years, AIOps (Artificial Intelligence for IT Operations) [55] approaches have been proposed to assist software practitioners process the large volumes of log data using data analytics and machine learning techniques [27, 38, 41]. Log data usually needs to be preserved for years to comply with legal regulations [57]. General compressors, like the *LZ77* compressor used in *gzip*, are typically used nowadays to compress log data to save storage space. Logging libraries [3, 7, 8] and log management tools (e.g., Splunk [1]) also use general compressors to compress log data for storage savings. However, such general compressors may not be optimized for log data. Listing 1 shows a snippet of log data produced by a software system.

```
2015-07-29 19:19:04,661-INFO-Received connection request /10.10.34.13:58116
2015-07-29 19:21:36,502-INFO-Received connection request /10.10.34.11:45957
2015-07-29 19:21:36,607-WARN-Interrupted while waiting for message on queue
2015-07-29 19:21:39,846-WARN-Connection broken for id 188978561024, my id= 1
2015-07-29 19:21:39,846-WARN-Interrupting SendWorker
2015-07-29 19:21:43,389-WARN-Interrupting SendWorker
2015-07-29 19:21:46,525-WARN-Connection broken for id 188978561024, my id= 1
2015-07-29 19:21:46,537-WARN-Send worker leaving thread
2015-07-29 19:21:46,728-INFO-Received connection request /10.10.34.13:58303
2015-07-29 19:21:49,960-INFO-Received connection request /10.10.34.12:48096
```

Listing 1: A snippet of log data.

As shown in Listing 1, each log line contains some fixed components (e.g., timestamps and log levels) and a free-form message body. In other words, log data is *semi-structured.* Some log lines are produced by the same logging statements in the source code. For example, the first and the second log lines in Listing 1 are produced by the same logging statement. Such log lines are identical except for some dynamic components (e.g., the IP addresses). Therefore, log data is *repetitive.* However, general compressors may not be designed to exploit these characteristics of log data, leading us to question the competence of general compressors in compressing log data.

In this paper, we study the use of general compressors on log data. We structured our study along three research questions (RQs) that are motivated as follows:

RQ1: *How well do general compressors compress log data?* Log data is usually compressed and archived before long-time storage. A common practice of log compression is applying general compressors on log files. However, because of the diversity of compressors and the various characteristics of log files, very different compression performance could be achieved for log compression. Thus, this RQ aims to understand the compression performance of general compressors on log data.

RQ2: *How much does the size of a log file impact its compression performance?* Modern log management tools usually split log files into smaller blocks before compressing each block separately. Different block sizes are used by different log management tools to split log files. In this RQ, we intend to understand how would the sizes of log files impact the compression performance of general compressors.

RQ3: *How do compression levels impact compression performance?* General compressors usually support configurable compression levels to make trade-offs between a faster compression/decompression speed and a higher compression ratio. These compressors usually specify a default compression level, which may not be optimal for log data. Therefore, in this RQ, we study the impact of compression levels on the performance of general compressors on log data.

In particular, we selected nine log files and a wide range of 12 general compressors for our study. We measured the performance of compressors through three measures: 1) the compression ratio, 2) the compression speed and 3) the decompression speed. We find that log data is much more repetitive than natural language data. Besides, the compressor with the highest compression ratio for natural language data is usually not the one for log data. Unfortunately, the compressors with the highest compression ratio for log data are often not the ones that are adopted by logging libraries or log management tools. More importantly, we find that the peak compression and decompression speeds of compressors on log data can be achieved with a small size of log data. As a common practice in log management tools (e.g., Splunk), dividing log data into small blocks and compressing them separately may not impair the compression ratio but rather improve the compression and decompression speeds. Finally, we observe that compression levels impact the compression ratio and the compression speed of log data more than the decompression speed. The default level is usually not the optimal level (measured by a combined compression performance score) for log compression.

Our study highlights the performance variances between compressing natural language and log data. Based on our findings, we encourage practitioners to find the optimal compressors and configure their compressors accordingly for their log data based on their usage scenarios. Our findings provide practitioners the insights of how to optimize the use of general compressors on log data, and call for future research on customized compression techniques for log data. For the reproducibility of our work, we also make our source code and the studied dataset publicly available[1]. The main contributions of the paper are as follows:

1) This paper makes a first attempt to understand the performance of general compressors on log data and the performance variances between compressing natural language and log data.
2) This paper investigates the repetition characteristics (e.g., local repetitiveness) of log data. Our findings provide insights for future work to develop customized compression approaches for log data.
3) This paper studies the impact of log sizes and compression levels on the performance of compressors on log data. Our findings provide a benchmark on log compression for both practitioners and researchers.

Paper organization. Section 2 and 3 describe the background and related work of log compression, respectively. Section 4 describes the experimental setup of our study. Section 5 present the design of our experiment that is guided by our three research questions. Section 6 presents our experiment results that provide answers to our research questions. Section 7 discusses the potential threats to the validity of our findings. Finally, Section 8 concludes the paper.

2 Background

In this section, we discuss the background of compressing log data from the perspectives of log management tools and log rolling.

2.1 Log Management Tools

Log management tools, such as Elasticsearch-Logstash-Kibana (ELK) stack [19] and Splunk [1], can assist practitioners in the processing and management of the rapid-growing log data. Log management tools usually support log parsing, storage, search, analysis, and visualization [5]. In order to save disk space, log management tools usually compress the stored log data. For example, Splunk uses *gzip* to compress the stored data while ELK supports *lz4* (for faster compression and decompression speeds) and *deflate* (for a higher compression ratio). In addition, log management tools usually divide the input log data into small blocks (or slices) and then apply compression on each of the blocks,

[1] Our replication package is available using the following link:
`https://queensuca-my.sharepoint.com/:f:/g/personal/18ky10_queensu_ca/EuI65h-Y0bJ0iukTY55RGx4BPX_-bGCgQantRkoMRAE_LA?e=IFhF9E`
Password: *SAILResearchLogCompression*
We will open the access to this package on a GitHub repository when the paper is accepted.

such that the compressed data could be decompressed and searched quickly (only the blocks containing the searched keywords need to be decompressed). For example, Splunk divides the input data into 128KB blocks and compresses each of them separately [15]. ELK by default splits log data into 16KB blocks. When a higher compression ratio is preferred, ELK splits log data into 60KB blocks[2].

However, it is not clear whether the compressors and block sizes supported by log management tools are optimal for the compression of log data. In this paper, we studied the performance of compressors on log data in choosing different compressors and block sizes. Our results can provide log management tool providers and users insights for improving the performance of compressors in log management tools.

2.2 Log Rolling

Log data usually grows very fast during system runtime [37]. To handle the fast-growth of log data, modern logging libraries (e.g, logback [7], log4j2 [3] and slf4j [8]) usually support the continuous archiving of log data, *a.k.a.*, log rolling. When a log file (e.g., `foo.log`) reaches a pre-defined size or time-based threshold, log file is automatically renamed (e.g., as `foo1.log`) and archived. A new log file with the same name (e.g., `foo.log`) is created for new log data. Such a process is called "rolling" or "rotation". For example, logback [7] supports log rolling based on a *TimeBasedRollingPolicy* or a *FixedWindowRollingPolicy*. The *TimeBasedRollingPolicy* triggers log rolling by time intervals (e.g., by day or by month), while the *FixedWindowRollingPolicy* triggers log rolling once a log file reaches a pre-defined maximum file size (i.e., the rolling size). However, it is unclear how rolling sizes impact the performance of compressors on log data.

In addition, developers can usually configure the rolling policy to automatically compress the rolled log files. Modern logging libraries usually support a few compressors for compressing the archived log files. For example, log4j only supports *gzip* and *zip* without external dependencies. Logback supports *gzip* and *zip*[3] and developers are expecting more alternative compressors[4]. However, the impact of using different compressors on the performance of compressing log data is not clear.

2.3 General Compressors

Logging libraries and log management tools commonly use general compressors (e.g., the *LZ77* compressor used in *gzip*) when compressing log data. These

[2] `http://lucene.apache.org/core/7_7_0/core/org/apache/lucene/codecs/lucene50/Lucene50StoredFieldsFormat.html`

[3] https://logback.qos.ch/manual/appenders.html

[4] https://jira.qos.ch/browse/LOGBACK-783

general compressors can be classified into three families: dictionary-based compressors, sorting-based compressors, and prediction-based compressors.

– **Dictionary-based compressors** use a dictionary to store the processed data and replace repeated occurrences of data with dictionary references.
– **Sorting-based compressors** use various strategies (e.g., move-to-front) to sort similar data together to increase the compression ratio.
– **Prediction-based compressors** use statistical models to predict the next token according to the context, thereby reducing the bits that are needed to encode the next token.

The wide usage of general compressors in log management tools and logging libraries motivates us to study the performance of general compressors on log data, and to what extent do the sizes of log files impact the compression performance. Our findings of this paper provide insights for log management tools and logging libraries to optimize their support for compressing log data.

3 Related Work

Prior work proposes various approaches to pre-process log data before applying general compressors, in order to improve the performance of compressors on log data. However, the computational cost of the extra pre-processing step usually prevents these approaches from being widely adopted in practice [54, 61]. We categorized these pre-processing approaches into four categories, namely delta encoding, bucketing, text replacement, and log transposition.

Delta encoding. Based on the observations that adjacent log lines tend to be similar, prior work proposes approaches to encode log lines using the delta to their preceding log line [14, 20, 61, 62]. For example, Balakrishnan et al. [14] propose a pre-processing approach for the system log data of the IBM Blue Gene/L supercomputer. Their approach compares a given log line with the previous log line and encodes only the differences, such that the given log line is encoded in a shorter version if the two log lines are similar. Their approach, when used together with general compressors, achieves an average of 28.3% improvement in compression ratio over general compressors.
Bucketing. Prior work clusters similar log lines into the same buckets before performing log compression [17, 20]. For example, Christensen et al. [17] cluster log messages into multiple buckets based on their textual similarity, then apply general compressors on these buckets. Their approach improves the compression ratio using *gzip* by 30% in an experiment of compressing Apache web server log.
Text replacement. Prior work proposes approaches that replace particular fields or common words/phrases in log data with shorter representations [53, 54, 61]. Otten et al. [54] propose a pre-processing approach that first converts timestamps and IP addresses into a shorter binary representation (e.g., 4 or 8 bytes for a timestamp; 3 to 11 bytes for an IP address); then replaces the semantic knowledge (i.e., common words and phrases) in log files

with unused ASCII characters. Their approach achieves an up to 32% improvement in compression ratio when used together with general compressors.
Log transposition. Prior work proposes approaches that parse each log line into fields (each field stores individual information like IP address and timestamp) and reshape log data such that the similar field values are placed together [18, 40, 48]. Such pre-processing approaches only work for log data with a fixed format. Mell et al. [48] propose a packing approach to improve the compression of structured log data. They parse log data into a matrix table where each log message is a row in the matrix. The matrix table is transposed to place similar tokens (i.e., the tokens of the same fields) closer to each other. Their pre-processing approach, when used together with *7zip_lzma*, achieves up to 21% improvement in compression ratio when compressing fixed-format log data (e.g., Microsoft Windows security log).

These pre-processing approaches usually consider log files of a single format or a few randomly selected log files, with random sizes. They typically use one or a few general compressors as benchmarks, without considering the impact of compression levels on compression performance. To the best of our knowledge, there exists no prior work that systematically studies how general compressors perform on log data. In this paper, we performed a comprehensive study of the performance of using general compressors on log data. Our results provide a benchmark for future work that aims to improve the performance of compressors on log data.

4 Experimental Setup

In this section, we present our experiment setup, including the preparation of our subject data, the selection of subject general compressors, and our experimental settings.

4.1 Subject Data

Software systems produce various types of log data. Given that there are so many different software systems (e.g., GitHub hosts millions of software projects), it is impossible to explore all the types of log data. Therefore, we focus on the log data that have been made public in prior work for log compression [11, 14, 17, 18, 20, 23, 24, 25, 40, 48, 53, 54, 56, 61, 62] and log analytics [28, 32, 42, 43, 51, 52, 63, 65, 66, 67, 73, 74, 75, 76, 78]. We followed a snowball literature review strategy [70] to find the related work on log compression and log analysis, in order to obtain a comprehensive view of the log data that is used in prior studies. We started our search on ACM Digital Library[5], IEEE Xplore Digital Library[6] and Springer Link[7]. To search for

[5] `https://dl.acm.org/`
[6] `https://ieeexplore.ieee.org/Xplore/home.jsp`
[7] `https://link.springer.com/`

Table 1: Our selected log data and natural language data.

	Data File	Raw size(GB)	#Templates	Static-dynamic ratio*	Description
Log Data	Access log	2	13	0.23	The access log is generated from a Tomcat web server. Similar access log is used in prior work for log compression. [54]
	Firewall log	2.92	11	0.46	The firewall log is used to track firewall activities in an operating system. The firewall log is used to detect frequent patterns in log data in prior work [25]
	HDFS log	1.6	149	0.66	The HDFS log is generated from a Hadoop cluster. It is used for log parsing in prior work [78].
	Liberty log	30	1034	1.18	The Liberty log is an aggregation of event log data on a supercomputer system. It is used for alert detection in previous studies [51, 52].
	Linux syslog	1.5	1702	1.27	The Linux syslog is generated from different applications running on a single machine. It is a centralized collection of log data from different applications (e.g., mail logs and cron logs). Syslog is used to analyze compression on log data in prior research [54, 61].
	Spark log	2.8	336	1.63	The Spark log is an aggregation of event log data. It contains log data from the Spark system and it is used in previous studies on log analysis and pattern extraction [28, 78].
	Thunderbird log	32	6404	2.25	The Thunderbird log is a combination of syslog from different machines. It is used to evaluate the performance of compressors on large-scale cluster log data in prior work [14, 40].
	Spirit log	38	1542	1.93	Similar to the Thunderbird log, Spirit log is also an aggregation of system log data from the Spirit supercomputer system. It is used for log analytics such as fault detection in previous studies [51, 63].
	Windows log	27	1842	0.86	The Windows log is produced by Windows 7's component based servicing (CBS), which records component installation and updating activities. It is used for log parsing in a prior study [78].
Natural Language (NL) Data	Gutenberg corpus	1.2	NA	NA	The Gutenberg corpus is a collection of over 3,000 English books. The Gutenberg corpus is widely used to evaluate the performance of a large collection of compressors on text compression in prior work [13, 46]. The Gutenberg corpus is also compared with software engineering data for their repetitiveness [26, 30].
	Wiki corpus	3.2	NA	NA	The Wikipedia corpus is data dump of English articles from Wikipedia. The Wikipedia corpus is widely used to evaluate the performance of different compressors on text compression in prior work [44].

* Static-dynamic ratio is calculated by dividing the size of static information (e.g., log templates) by the size of the dynamic information (e.g., parameters in log templates). We leverage a state-of-the-art log abstraction tool [78] to identify the static and dynamic information from each of the studied 1GB log files.

both prior research on log compression and log analysis, we performed three queries with keywords "log compression", "log file compression" and "log analysis". As a result of our literature review, we collected 25 log files from prior research [11, 14, 17, 18, 20, 22, 23, 24, 25, 28, 32, 40, 42, 43, 51, 52, 53, 54, 56, 61, 62, 63, 66, 67, 73, 74, 75, 76, 78].

From the collected log files, we focus on the ones that have a size of at least 1GB. We did not include smaller log files as we need a fair comparison among different log files and between log files and natural language corpora (as mentioned below, we truncated all studied log files and natural language corpora into the same size of 1 GB). Besides, as log data is printed according to a time sequence, the log messages in a small-sized log file might only record the events in a short time slot, which is not sufficient to represent the real distribution of log messages. As a result, we selected the following nine log files, i.e., an Access log file, a Firewall log file, a HDFS log file, a Liberty log file, a Linux syslog file, a Spark log file, a Spirit log file, a Thunderbird log file and a Windows log file.

Table 1 presents an overview of our selected log files and the descriptions of their formats. For each log file, we also measured the number of unique templates and the ratio between the static and dynamic content, using a state-of-the-art log parsing tool named *Drain* [78]. Such information can be useful for further understanding the characteristics of the studied log files and the performance of general compressors on these log files. In order to avoid the bias caused by different log sizes, we truncated each log file from the beginning so that each has the same size of 1GB.

In addition to log data, we also selected two natural language corpora as our baseline data: the Gutenberg corpus[8] and the Wikipedia corpus[9]. The Wikipedia corpus has been used as the subject data for text compression benchmarks [46]. The Gutenberg corpus is widely used as a baseline to compare the repetitiveness of software engineering data with [26, 30]. Furthermore, the Gutenberg corpus is used to evaluate the performance of compressors on natural language data in prior work [13, 46]. Since the Wikipedia corpus is stored in XML format, we cleaned the data set by filtering out XML tags and external links. We also truncated each natural language corpus from the beginning so that each has the same size of 1GB.

The nine log files and the two natural language files, each with the size of 1GB, are used as the subject data in our experiments.

4.2 Subject General Compressors

In this paper, we considered a wide variety of general compressors in our experiments. In particular, we selected the general compressors that are used in two large text compression benchmarks [4, 6]. From each family of compressors (c.f. Section 2.3), we selected a few representative compressors (e.g., those used in prior studies). In total, we selected 12 general compressors across the three families. For each compressor, we chose an implementation that is publicly available and free to use for our experiments. Table 2 describes the selected compressors and the corresponding implementations[10].

4.3 Experimental Environment

Our experiments were performed on a server running Ubuntu 16.04.5 with 80 cores of Intel Xeon E7-4780 @ 2.4GHz CPU and 504GB total memory. The server provides an SSD storage of 6.8TB. We installed all the compressors that are listed in Table 2 and their dependent libraries on the server. We used *psutil*[11] to monitor the compression and decompression processes, in order to measure the CPU time spent on compressing and decompressing each file. We

[8] `https://web.eecs.umich.edu/~lahiri/gutenberg_dataset.html`
[9] `https://dumps.wikimedia.org/enwiki/`
[10] We also included our used compressor implementations in our replication package.
[11] `https://github.com/giampaolo/psutil`

Table 2: The selected general compressors in our study.

Family	Algorithm	Implementation	Description
Dictionary-based	LZ77	gzip[1]	This algorithm replaces repetitive inputs by referring to the previous occurrences stored in a dictionary.
	LZW	compress[2]	This algorithm encodes data by dynamically building a dictionary. LZW first creates the dictionary of 1-byte symbols, then adds the index of the longest match to the dictionary.
	LZ4	lz4[3]	LZ4 has a similar algorithm as LZ77 but it uses a fixed and byte-oriented encoding. LZ4 features extremely fast compression and decompression speeds.
	LZMA	7zip_lzma[4]	LZMA is a variant of LZ77. LZMA features high compression by using varying dictionary sizes (up to 4 GB).
	LZSS	quickLZ[5]	LZSS is similar to LZ77, except that LZSS ignores encoding a string if the dictionary reference is not shorter than the string, while LZ77 encodes every input string. LZSS features high speed rather than high compression ratio.
Sorting-based	SR/MTF	sr2[6]	Symbol Ranking, also known as Move To Front, transforms each symbol in the data and replace its index in a *recently used symbols* stack. The core idea behind this algorithm is that the recently seen symbols are more likely to occur again.
	BWT	bzip2[7]	BWT first sorts each character by context (put characters with similar context together), then compress the transformed data by the order-0 symbol ranking algorithm.
	ST	szip[8]	ST is based on a modified BWT algorithm. ST uses byte-oriented encoding and a blocksort-based prediction model. BWT performs better for larger files.
Prediction-based	PPMD	7zip_ppmd[4]	PPMD is a statistical compression algorithm based on context modeling and prediction. PPMD uses an order-n Markov model that predicts the n$^{\text{th}}$ token based on context (i.e., prior $n-1^{\text{th}}$ tokens).
	DMC	ocamyd[9]	Similar to PPMD, DMC also predicts next tokens based on the context. DMC is based on bit-level predictions, while PPMD is based on byte-level predictions.
	CM	zpaq[10]	CM is based on next-token predictions that combine two or more statistical models, in order to have a higher prediction accuracy than single model.
	CTW	ctw[11]	CTW is based on bit-level statistical modeling. CTW combines the predictions of variable order Markov models. CTW yields high compression ratio but slow speed in the context of text compression.

[1] https://www.gnu.org/software/gzip/
[2] http://manpages.ubuntu.com/manpages/bionic/man1/compress.1.html
[3] https://lz4.github.io/lz4/
[4] https://www.7-zip.org/
[5] http://www.quicklz.com/
[6] http://mattmahoney.net/dc/index.html#sr2
[7] http://www.bzip.org/
[8] http://www.compressconsult.com/szip/
[9] http://www.geocities.ws/ocamyd/
[10] http://mattmahoney.net/dc/zpaq.html
[11] https://web.archive.org/web/20150302190939/http://www.ele.tue.nl/ctw/

used CPU time instead of elapsed time in order to reduce the noise caused by other processes running in the same environment.

5 Experimental Design

We study the performance of general compressors on log data along three research questions. In this section, we present the motivation and our approach for answering each of the research questions. Before presenting our RQs, we perform a preliminary analysis of the characteristics of log data.

Preliminary analysis: How repetitive are log and natural language files?

Motivation

Lossless data compression typically exploits repetitive information in the data, then represents such repetitiveness of data with a lower number of bits [58]. Thus, an understanding of the repetitiveness of the data is of great importance before evaluating the compression. In the preliminary study, we investigate the repetitiveness of different log data. In order to have a baseline to compare with, we also evaluate the repetitiveness of natural language data, similar to prior research [26, 30].

Approach

We used entropy to measure the repetitiveness of log data. Shannon's entropy [60] (or entropy) is used to measure the amount of information that is contained in an information source (e.g., a text file). Entropy is calculated as $H = -\sum_{i=1}^{n} p(i) log_2 p(i)$, where $p(i)$ is the probability of a possible state of the information source (i.e., the possibilities of a sequence of characters in English language [60]). The more random (i.e., less repetitive) the information source, the higher the entropy value.

Entropy is relative to the model (e.g., n-gram model) that captures the probability distribution of the information source [30, 60]. We used an n-gram model to capture the probability distribution of our data. An n-gram model predicts the probability distribution of the next item in a sequence based on an order-n Markov model which assumes that the i^{th} item in the sequence can be predicted from the prior $n-1$ items [35]:

$$p(m_i|m_{i-1}, m_{i-2}, ..., m_{i-n+1}) = \frac{count(m_i, m_{i-1}, m_{i-2}, ..., m_{i-n+1})}{count(m_{i-1}, m_{i-2}, ..., m_{i-n+1})} \quad (1)$$

Based on the n-gram model, the entropy of a sequence of tokens is calculated as:

$$H = -\frac{1}{N}\sum_{i=1}^{N} \log p(m_i|m_{i-1}, m_{i-2}, ..., m_{i-n+1}) \quad (2)$$

where N is the number of tokens in the sequence.

Specifically, inspired by prior work [26, 30], we used *cross-entropy* to measure the repetitiveness of our data. For each log or natural language file, we used ten-fold cross-validation to calculate the cross-entropy value: we used nine folds to train an n-gram model and test the n-gram model on the remaining fold. We repeated the process ten times and calculate the average entropy value. For each log/natural language data, we calculated the average cross-entropy value over ten randomly selected 512MB blocks. There might be some n-gram sequences that appear in the testing data while unseen in the training data, which would lead to a probability estimation of zero followed by an infinite entropy value (when $p(i) = 0$ then $log_2 p(i) = -\infty$). In order to avoid such infinite entropy values while still producing entropy values with statistical rigor, we used a Modified Kneser-Ney smoothing approach [36], which is proven to achieve robust entropy estimations when dealing with software corpus [26, 30]. In the rest of the paper, we used *cross-entropy* interchangeably with *entropy.*

RQ1: How well do general compressors compress log data?

Motivation

Compressing log data is important in log management. For example, log management tools such as ELK and Splunk usually compress log data before storage. When a specific performance measure is more preferred (e.g., when compression or decompression speed is favored over compression ratio), the best performing compressor for different measures may be different. However, there exists no prior work that systematically studies the performance of general compressors on log data. Our answers to this research question can assist software practitioners and logging library/log management tool providers in choosing appropriate compressors under different usage scenarios.

Approach

Evaluating compression performance.

We evaluated the performance of general compressors on a variety of log formats. We used our selected compressors to compress/decompress our selected files and measure the corresponding compression/decompression performance. For each compressor, we used the default compression level. We used the following three measures to evaluate compression performance:

– **Compression ratio:** the ratio of the size of the original (uncompressed) data to the size of the compressed data. The compression ratio indicates how much the size of a file can be reduced by compression.
– **Compression speed:** the size of the original (uncompressed) data divided by the compression time. The compression speed shows how fast a piece of data can be compressed.

– **Decompression speed:** the size of the original (uncompressed) data divided by the decompression time. The decompression speed shows how fast a piece of compressed data can be decompressed.

Using statistical analyses to compare the performance of compressors.

One may not be able to draw a conclusion that one compressor is better than another from a single run of compression on one instance of subject data, since the results are likely to be biased by the peculiarity of the data instance. To reduce the bias and make the results more generalizable, when comparing different compressors, we randomly selected ten 512MB blocks from each log/-natural language file and run the compression/decompression experiments for each block. Then, we used Scott-Knott clustering to group the compressors into statistically distinct groups based on each performance measure (compression raito, compression speed, and decompression speed). The Scott-Knott algorithm hierarchically clusters the compressors by statistically measuring the difference between the performance measures of the compressors [31, 59]. As a result, two compressors in different groups will have statistically different compression performance measures, while two compressors within the same group will not.

RQ2: How much does the size of a log file impact its compression performance?

Motivation

Prior research on text compression finds that files with larger sizes may be compressed with a higher compression ratio, due to the higher possibility of having repetitiveness [6]. However, the practitioners in log analysis and log management favor the contrary, i.e., log data is more suitable to be compressed in smaller sizes in practice. The reason is that retrieving a log line from compressed log data requires the decompression of half of the data on average [40]. For log analysis applications that need to retrieve individual log lines frequently, it is inefficient to compress log data in large sizes and retrieve information after decompressing the entire data. To improve the efficiency of retrieving information from compressed log data, the large volume of log data is often split into smaller blocks and compressed separately. Only the blocks that contain the retrieved information need to be decompressed. For example, ELK splits data into 16KB or 60KB blocks and compresses such small blocks separately. Similarly, Splunk splits data into 128KB blocks. However, it is not clear whether the small data sizes bring negative impact on the performance of general compressors. In this research question, we investigate the impact of the size of a log file on the performance of general compressors.

Approach

First, we analyzed the impact of the size of a log file on the entropy values, in order to understand the repetitiveness at varying sizes of log files. We use the scope of repetition, which is an indicator of how much the block size influences the repetitiveness, to examine whether the repeated contents are close to each other in the files. Then, we performed experiments to evaluate the performance of general compressors on varying sizes of log files. Based on surveying prior literature on log compression [14, 17, 18, 20, 48, 53, 54, 61], we selected three widely considered compressors from different compressor families: the *LZ77* compressor from the dictionary-based family, the *BWT* compressor from the sorting-based family, and the *PPMD* compressor from the prediction-based family.

Randomly extracting data with varying sizes.

In order to investigate the impact of log sizes on the performance of general compressors on log data, we randomly extracted data of different sizes from our subject data. From each of the 1GB log file, we randomly picked a starting point in the file and read a data block of a given size (e.g., 128KB) starting from that random point. We started by extracting the data with 1KB, and we doubled the size until it reaches 512MB. We ensured that the size from the randomly picked point to the end of the file is not smaller than the given data size to be extracted. For each log and natural language data with each size, we repeated the process ten times to avoid the random bias caused by using a single data block. We also used the same approach to randomly extract natural language data with varying sizes. The random sampling process makes our results more generalizable to the selected data compared to showing the results on a single data block.

Calculating entropy of varying sizes of data.

For every log and natural language data with each size, we used the approach discussed in the preliminary analysis to calculate its entropy value. The entropy values for different sizes of a file can indicate the scope of repetition of that file. For example, a smaller entropy value for a certain file size indicates a higher repetitiveness at that file size.

Measuring compression performance for varying sizes of data.

For every log and natural language data with each size, we used the approach discussed in RQ1 to measure the compression ratio, compression speed, and decompression speed of the three selected compressors. When evaluating the CPU time used for compression and decompression, we realized that it was difficult to capture the precise time taken to compress/decompress small files (e.g., less than one second). Thus, we repeated the process of compressing/decompressing small files multiple times and measure the total time taken for these repetitions. Then, we calculated the average time taken for each repetition. The number of repetitions is determined by Equation 3:

$$\text{Repetitions} = \beta \times \log_2(\frac{S_0}{S_i}) \tag{3}$$

where S_0 and S_i are the original file size (i.e., 1GB) and the processed file size, respectively, and β is a coefficient that accounts for the difference of compression/decompression speed between compressors. For example, as the speed of the *LZ77* compressor is much faster than the BWT compressor and the *PPMD* compressor, we set a higher β value for the *LZ77* compressor (i.e., more repetitions). The intuition is that the files compressed faster need more repetitions to capture the compression speed.

RQ3: How do compression levels impact compression performance?

Motivation

Modern compressors usually support configurable compression levels (i.e., one to nine) to make tradeoffs between high compression ratios and fast compression/decompression speed. A higher compression level usually provides a higher compression ratio but a slower compression speed. A lower compression level usually provides a faster compression speed but a lower compression ratio. General compressors usually specify a default compression level, while the default compression levels may not be optimal for log data. Therefore, in this research question, we study the impact of compression levels on the performance of general compressors on log data. Our findings provide practitioners with guidance on choosing a proper compression level of compressors for better performance. Log management tools and logging libraries providers can also learn from our findings to improve the performance of their supported compressors.

Approach

Measuring compression performance under different compression levels.

General compressors usually support compression levels from one to nine. A higher compression level usually indicates a higher compression ratio and a lower compression speed. All the selected compressors support the nine compression levels[12]. For each file, we randomly chose ten 512MB blocks and use the three compressors to compress each block separately. The sizes of blocks before compression are set to be the same. Thus, the size after compression of each block (higher or lower compression ratio) will only be impacted by changing compression levels. We repeated our experiments nine times using the nine different compression levels. The decompression step can automatically recognize the used compression levels of the compressed files. Thus, we did not configure specific levels for decompression. We measured the compression ratio, compression speed, and decompression speed under different compression levels.

[12] The selected compression tools *gzip*, *bzip2*, and *7zip_ppmd* implement the nine compression levels of LZ77, BWT, and PPMD, respectively.

Calculating a combined compression performance score.

Compression levels provide various tradeoffs between compression ratio and speed. In order to understand the overall performance of our selected compressors under different levels, we used a combined compression performance score [9, 12] that integrates compression ratio and speed. The combined compression performance score (C_Score) is defined as follows:

$$C\_Score = log_{10}(2^{\frac{\frac{size_x}{size_{top}}-1}{0.1}} \times time_x) \tag{4}$$

where $size_{top}$ represents the smallest compressed data size achieved by various levels of a compressor, $size_x$ is the compressed data size achieved by the current compression level, and $time_x$ is the total time spent on compression and decompression with the current compression level.

6 Experiment Results

In this section, we first present the results of our preliminary analysis. Then, we present the results of our experiments for answering our three research questions.

Preliminary analysis: How repetitive are log and natural language files?

Result

Figure 1 shows the average cross-entropy values of the log and natural language data using a 5-gram model. We measured the entropy values using gram sizes of one to ten, and we found that the entropy values decrease as the gram size increases, which is because longer sequences of context information can usually provide a better estimation of the next token. However, the entropy values stabilize as the gram size reaches five. Figure 1 shows that the entropy values of log data (i.e., 0.21 to 0.61 using 5-gram models) are much lower than the entropy values of natural language data (e.g., 1.37 to 1.71 using 5-gram models). The lower entropy values from log data validate that log data is much more repetitive than natural language data. We find that some specific log data (e.g., HDFS log) may have a relatively higher entropy when compared to other log data. However, after comparing the number of unique log templates and the ratio between the static and dynamic contents among the studied log data, we cannot find a clear relationship between these characteristics and the entropy values. Therefore, future research may consider other factors to explain the variance in entropy values of log data. In general, we observe from our preliminary analysis that **log data is highly repetitive, showing a much lower entropy than natural language data using n-gram models.**

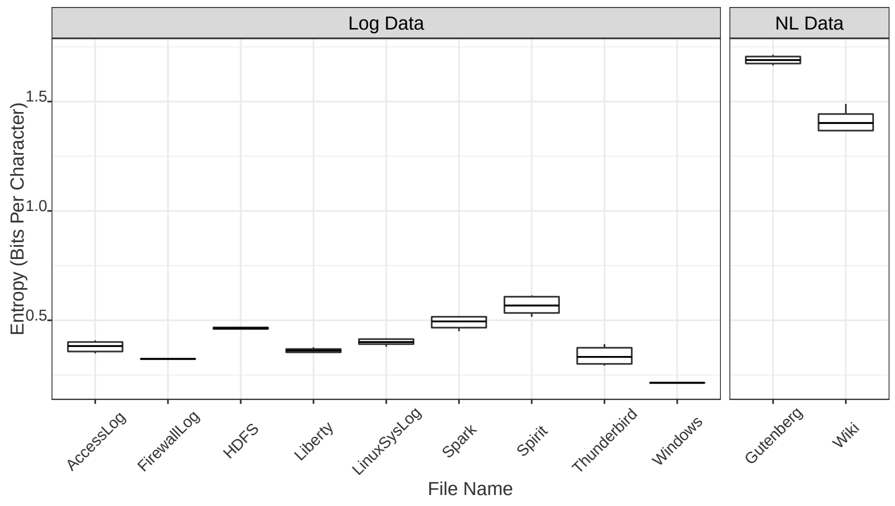

Fig. 1: Cross-entropy of the log and natural language (NL) data of 512M size. For each file, we calculated the cross-entropy of ten random blocks. The line inside the box indicates the mean cross-entropy over the ten random blocks.

RQ1: How well do general compressors compress log data?

Result

Figure 2 compares the compression ratio, compression speed, and decompression speed of different compressors on the selected log and natural language data. Table 3 shows the results of using Scott-Knott to cluster the compressors based on their performance. For each compression performance measure (i.e., compression ratio, compression speed, and decompression speed), we highlighted the top three compressors for each data. General compressors achieve up to 17.07 to 249.44 compression ratio for different log data. In comparison, general compressors only achieve up to 4.62 to 5.00 compression ratio for natural language data. In addition, compressing/decompressing log data is faster than compressing/decompressing natural language data. The higher compression ratio and faster compression/decompression speed could be explained by the higher repetitiveness in log data. Taking the dictionary-based compressors (e.g., *LZ77* compressor) as an example, the more repetitive the data, the smaller the dictionary for storing and retrieving repetitive information, thus the higher compression ratio and faster compression/decompression speed [50].

Finding 1.1: Log data is compressed faster and gets a higher compression ratio than natural language data.

As the winner of the Hutter Prize [2, 10], the prediction-based compressor *CM* achieves the highest compression ratio for compressing natural language data (cf. Table 3). However, it may not achieve the best compression ratio

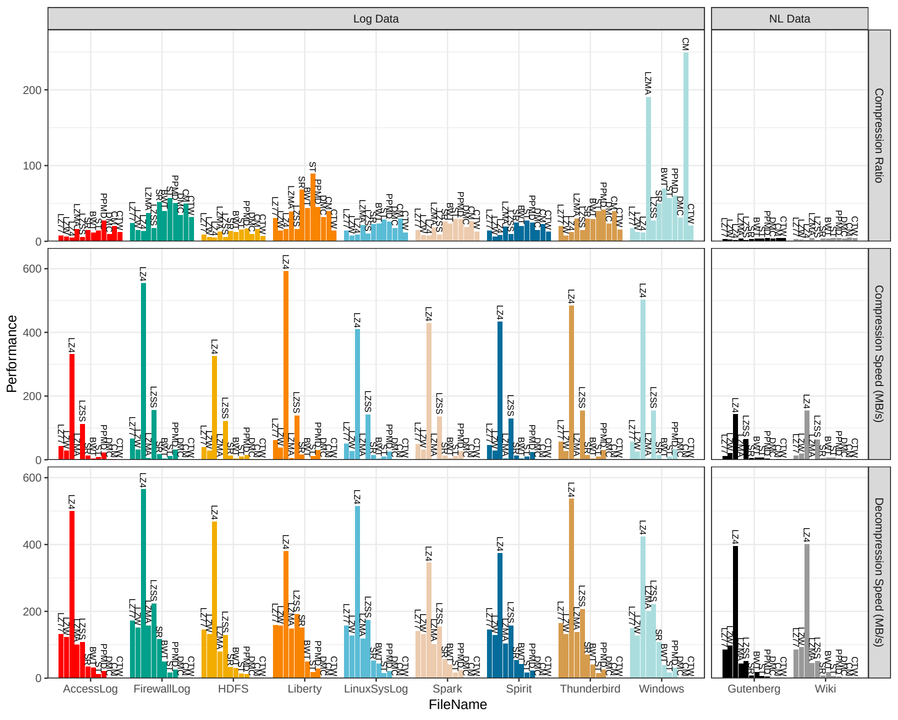

Fig. 2: Comparison of compression performance (compression ratio, compression speed, and decompression speed).

for log data. In fact, the compressor with the highest compression ratio varies across different log data. In particular, the *CM* compressor achieves the highest compression ratio for the Windows log, Linux syslog and the Thunderbird log. The *PPMD* compressor achieves the highest compression ratio for the Access log, the HDFS log, the Spark log and the Thunderbird log. The *ST* compressor achieves the highest compression ratio for the majority of log data except for Access log, HDFS log and Windows log. However, **neither the *CM*, the *PPMD* nor the *ST* compressors are adopted in existing logging libraries or log management tools as their compressors.** The mostly used compressors, i.e., the *LZ77* and the *LZ4* compressors, are only ranked 4 to 10 and 5 to 12 in terms of compression ratio, respectively.

Finding 1.2: The compressor with the highest compression ratio for natural language data (e.g., the *CM* compressor) is usually not the one for log data. The compressor with the highest compression ratio for one log format may not be the one for another log format either.

The compression and decompression speeds of the *LZ4* compressor are 6.6 to 8.5 and 1.4 to 2.8 times faster than the most commonly used compressor *LZ77* (based on which *gzip* is implemented), respectively, while the compres-

Table 3: Compressors ranks from Scott-Knott clustering. We used shading to highlight the top three compressors for each file in terms of different compression performance measures.

Compression Performance Measures	FileName	LZ77	LZW	LZ4	LZMA	LZSS	SR	BWT	ST	PPMD	DMC	CM	CTW
CompressionRatio	AccessLog	7	8	9	3	8	4	5	4	1	6	2	5
	FirewallLog	9	11	12	6	10	2	5	1	3	7	4	8
	HDFS	6	8	8	5	8	4	5	3	1	6	2	7
	Liberty	4	5	5	3	5	2	3	1	3	4	3	5
	LinuxSysLog	5	7	7	3	6	3	3	1	2	4	1	6
	Spark	4	5	5	3	5	2	2	1	1	3	2	4
	Spirit	5	9	8	4	7	2	4	1	2	5	3	6
	Thunderbird	4	7	6	2	5	2	2	1	1	3	1	5
	Windows	10	11	11	2	8	6	3	5	4	7	1	9
	Gutenberg	8	9	11	4	10	7	5	4	2	6	1	3
	Wiki	6	7	8	3	8	5	4	4	2	5	1	2
Average CompressionRatio	Log	6.0	7.9	7.9	3.4	6.9	3.0	3.6	2.0	2.0	5.0	2.1	6.1
	NL	7.0	8.0	9.5	3.5	9.0	6.0	4.5	4.0	2.0	5.5	1.0	2.5
CompressionSpeed	AccessLog	3	4	1	8	2	6	8	7	5	9	9	9
	FirewallLog	3	4	1	7	2	5	7	6	4	7	7	7
	HDFS	3	4	1	9	2	6	8	7	5	9	9	9
	Liberty	3	4	1	6	2	5	6	5	4	6	6	6
	LinuxSysLog	3	4	1	7	2	5	7	6	4	7	7	7
	Spark	3	4	1	6	2	5	6	5	4	6	6	6
	Spirit	3	4	1	6	2	5	6	5	4	6	6	6
	Thunderbird	3	4	1	6	2	5	6	5	4	6	6	6
	Windows	3	5	1	8	2	6	8	7	4	8	8	8
	Gutenberg	4	3	1	9	2	6	5	6	7	8	8	9
	Wiki	4	3	1	7	2	5	5	5	6	7	7	7
DecompressionSpeed	AccessLog	2	3	1	5	4	6	6	8	7	9	9	9
	FirewallLog	3	4	1	4	2	5	6	8	7	9	9	9
	HDFS	2	3	1	4	3	5	5	6	6	7	7	7
	Liberty	3	3	1	4	2	4	5	7	6	8	8	8
	LinuxSysLog	3	4	1	5	2	6	7	9	8	10	10	10
	Spark	3	4	1	5	2	6	7	9	8	10	10	10
	Spirit	3	4	1	5	2	6	7	8	8	9	9	9
	Thunderbird	3	4	1	4	2	5	6	8	7	9	9	9
	Windows	4	5	1	3	2	6	7	8	7	9	9	9
	Gutenberg	3	2	1	5	4	7	6	7	7	8	8	8
	Wiki	3	2	1	5	4	7	6	7	7	8	8	8

sion ratio of the *LZ4* compressor is 33.0% to 48.2% smaller. As shown in Figure 2 and Table 3, in general, the dictionary-based compressors achieve faster compression and decompression speeds than the sorting-based and the prediction-based compressors. The top fastest compressors *LZ4*, *LZSS* and *LZ77* are all dictionary-based compressors. In particular, as the default compressor of ELK, the *LZ4* compressor achieves much faster compression and decompression speeds than any other compressors (including other dictionary-based compressors) for both log and natural language data. On the other hand, all the dictionary-based compressors have rather low compression ratios.

Finding 1.3: Although having low compression ratios, the dictionary-based compressors achieve the fastest compression and decompression speeds for log data.

Discussion

Qualitative examination of compression results

As shown in Table 3, the compressors have different compression ratios on different log data. To further understand the relationship between the peculiarities of the considered compressors and the corresponding compression performance, we particularly examine our results in three folds as follows.

1) Some compressors achieve the most different compression ratios between log and natural language data. We find that the compression ratio of the *CTW* and *SR* compressors differs the most between compressing log data and compressing natural language data. The *CTW* compressor achieves an average rank of 2.5 in terms of compression ratio for compressing natural language data, but the average rank drops to 6.1 when compressing log data. In contrast, the average rank of the *SR* compressor in terms of compression ratio is 3.0 when compressing log data and it drops to 6.0 when compressing natural language data. The *CTW* compressor uses the Context Tree Weighting algorithm to predict and encode the next symbols in the data [69]. The Context Tree Weighting algorithm leverages all preceding symbols as its context to predict the next symbol. However, compared to natural language data, log data is usually locally repetitive (see Figure 4 and corresponding discussions in RQ2). Thus, in contrast to natural language data, using a larger context for log data may not produce better results than using a smaller context, which explains why *CTW* achieves a lower rank when compressing log data than compressing natural language data. The *SR* compressor is based on symbol ranking, which converts the data sequence by maintaining the order of the symbols from the newest to oldest [21]. The idea behind symbol ranking is that the most recently seen symbols are most likely to re-occur [45]. As discussed in RQ2, log data is locally repetitive (i.e., the recently seen symbols are likely to re-occur), the *SR* compressor tends to achieve a better rank in compressing log data than compressing natural language data.

2) Some compressors are more stable in compression ratios than other compressors. Furthermore, we observe that some compressors (e.g., *CM* and *PPMD*) are more stable in compressing log data than other compressors (e.g., *LZ4*, *LZ77* and *LZW*). The *PPMD* compressor predicts the next byte based on the longest seen context [45]. This compressor considers the context within the whole data. While the *LZ77* based compressors are based on a sliding window, which refers to a new character to its previous occurrence. However, if the distance between those two characters exceeds the sliding windows size, the previous occurrence is not matched. Compared to the *PPMD* compressor, the *LZ77* compressor only processes part of the data sequentially. Thus, the *PPMD* compressor could achieve a higher compression ratio and it is more stable than the *LZ77* compressor.

3) Some log files can be better compressed than other log files. From Figure 2, we observe that some log files (e.g., the Windows log, the Firewall log, the Thunderbird log, and the Liberty log) are usually better compressed than other log files. In particular, the *CM* and *LZMA* compressors achieve extremely high compression ratio on the Windows log (i.e., up to a compression ratio of 249.33). As shown in Figure 1, these log files have lower entropy values (i.e., more repetitive) than others. By analyzing the characteristics of the studied log files (i.e., the number of unique log templates and the static-dynamic ratios, as shown in Table 1), we did not observe a clear relationship between such characteristics and the compression ratios. Thus, the difference between the compression ratios may be due to some other factors (e.g., the log templates of the highly compressed log files are relatively more similar to each other).

The choice of compressors should depend on the usage scenarios.

Our findings and the qualitative analysis above show that there exists no best compressor for all scenarios. The widely used *LZ77* compressor is a good tradeoff between compression ratio, compression speed, and decompression speed (cf., Figure 2 and Table 3). However, such a trade-off may not always be the optimal choice. In fact, due to the variety of compression performance, the choice of compressors should depend on the specific usage scenarios. In previous research, Otten et al. [54] summarized four common scenarios when compressing log data. We discuss the compressors that are most applicable to each scenario:

1) **Collecting log data to a central location for analysis**: This scenario prefers a high compression ratio without considering compression and decompression time. The *PPMD* compressor could be used to compress log data. We find that *PPMD* compressors is slow (e.g., 66.2% to 71.3% slower than the *LZ77* compressor) in compressing natural language data. However, the *PPMD* compressor is only 41.5% to 60.6% slower than the *LZ77* compressor when compressing log data. In fact, using the *PPMD* compressor to compress log data is faster than using the *LZ77* compressor to compress natural language data. For example, the *LZ77* compressor has a compression speed of 13.8MB/s when compressing the Wikipedia data, while the *PPMD* compressor only has a speed of 3.97MB/s to compress the same data file. However, the *PPMD* compressor has a compression speed of 15.64MB/s to 33.33MB/s when compressing log data, which is even faster than compressing natural language data using the *LZ77* compressor.

2) **Real-time monitoring**: This scenario features a minimum point-to-point time usage with fewer resources (e.g., memory, bandwidth) usage. When log data needs to be compressed and transferred in a timely manner, the compressor with a faster speed and acceptable compression ratio is preferred. In particular, the *LZSS* compressor could be used instead of the *LZ77* compressor.

3) **Quick access (storing it compressed and later decompressing it for analysis)**: This scenario prefers high decompression speed over compression speed. Furthermore, little bandwidth usage (i.e., high compression ratio) is also preferred in this scenario. For example, when log data is gradually compressed while being produced over time, the *LZMA* compressor could be selected over the *LZ77* compressor. The *LZMA* compressor achieves a proper decompression speed (ranges from rank 3 to rank 5 at different log data) with a relatively high compression ratio.

4) **Low system-time-usage for compression and decompression**: This scenario features fast compression and decompression speed, while not considering the performance of compression ratio. The *LZ4* achieves the top

compression speed and decompression speed on all the studied files. Thus, it is the most applicable compressor for this scenario.

***N*-gram model-based entropy value cannot capture the compression limits of log data.**

In Figure 3, for each file, we compared the compression ratios of different compressors with the 5-gram model-based entropy value. We used the 5-gram model because we observe that the entropy values stabilize after a gram size of 5. Here we converted the compression ratio into the unit of bits per character (i.e., $\frac{1}{\text{Compression Ratio}} * 8$ bits/character). N-gram models are widely used to model natural languages data and to capture the compression limits of natural language data. However, n-gram models fail to capture the compression limits of log data. Figure 3 shows that the compressor with the highest compression ratio can achieve better compression ratios than the entropy values, which indicates that the compressor with the highest compression ratio can capture the repetitiveness in log data better than n-gram models. One possible reason is that the diversity of dynamic information influences the prediction accuracy of the n-gram model. Although log data is more repetitive, the repetitiveness is mostly contributed by semantic text inside log data. For example, considering the logging statement $log.info("ProductID\ is\ " +\ id)$, we notice that the dynamic information id is always followed by the static text. The diverse value of such dynamic information makes it hard to predict from the prior static text. Due to such characteristics of log data, the n-gram model may not accurately estimate the entropy values for log data.

On the other hand, the entropy values and compression ratios are highly correlated across different files. In fact, the entropy values and the compression ratios have a Spearman correlation of 0.87. Practitioners may use the entropy values to estimate the rank of compression ratios of different log data, instead of actually apply the compression.

Log data is compressed faster than natural language data using general compressors with a higher compression ratio. The compressor with the highest compression ratio for natural language data may not be the one for log data; while the compressor with the highest compression ratio for log data may not be adopted by logging libraries and log management tools in practice. Since there exists no compressor that has the optimal performance in all measures, one should choose optimal compressors according to their usage scenarios.

RQ2: How much does the size of a log file impact its compression performance?

Result

Figure 4 shows the entropy values of the selected log and natural language data over varying data sizes. Natural language data has an increasing trend of repetitiveness (decreasing entropy) when data size increases, which agrees

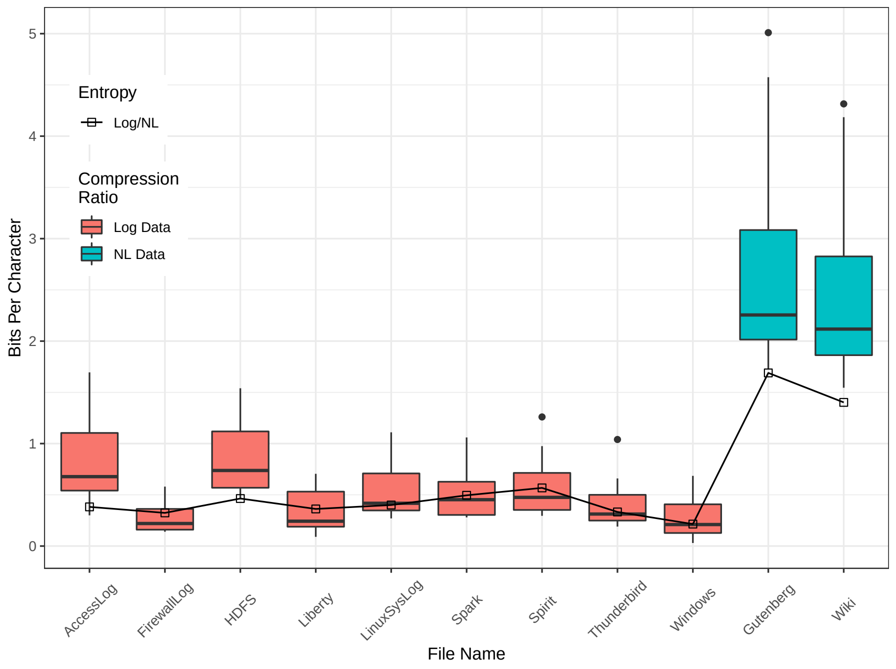

Fig. 3: Comparing the compression ratios of different compressors and the 5-gram model-based entropy values (5-gram). The entropy values and the best compression ratios have a Spearman correlation of 0.87.

with the prior research finding [6]. Hence, natural language data reaches the lowest entropy values at large data sizes. On the other hand, log data reaches the lowest entropy values at the sizes of 16KB to 8MB, then the entropy values start to increase with larger data sizes. Such results show that log data can be most repetitive at a certain small data size. The lowest entropy values with a certain small sizes of log data indicate that log data has stronger local repetitiveness (i.e., the scope of repetition at small file sizes) and weaker global repetitiveness (i.e., the scope of repetition at large file sizes). As log entries are produced by logging statements in the source code, log entries within a small block are more likely to share the same information (e.g., dynamic information). Future log compression approaches should consider the local repetitiveness of log data to improve the performance of compressors on log data.

In addition, we observe that the access log has a larger scope of repetition compared to other log data (see Figure 4), which may be explained by the larger amount of dynamic information in the access log. Dynamic information is usually not repeated locally; in comparison, the static information in log data tends to be locally repeated [61].

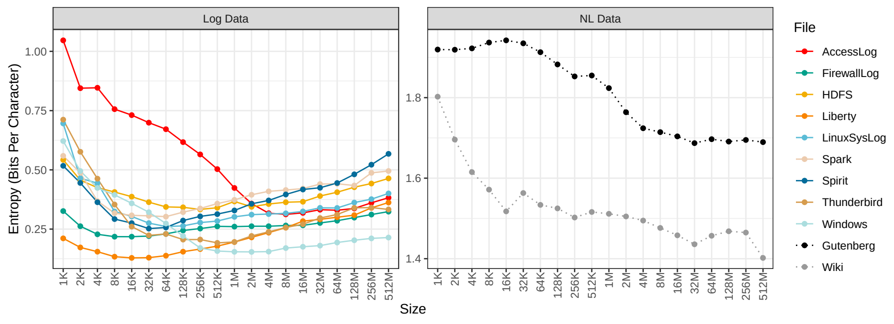

Fig. 4: Cross-entropy from the 5-gram model on log and natural language (NL) data for different block sizes.

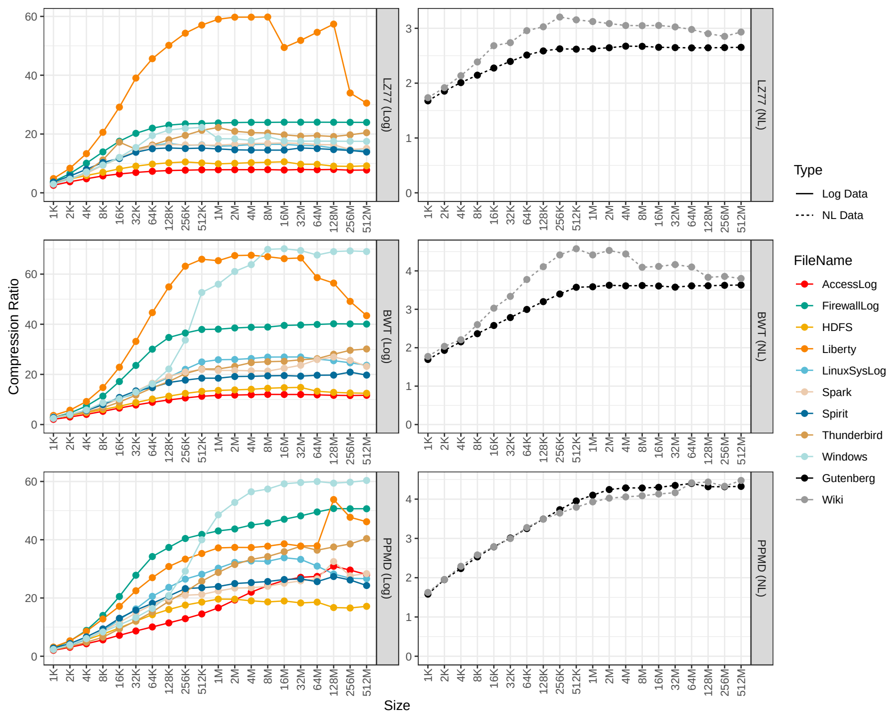

Fig. 5: Compression ratio of log and natural language (NL) data at different block sizes.

Finding 2.1: Log data with a certain small size has the highest repetitiveness, i.e., log data has stronger local repetitiveness and weaker global repetitiveness.

Figure 5 shows that overall, the compression ratio increases as the data size increases. However, the compression ratio of log data reaches saturation when

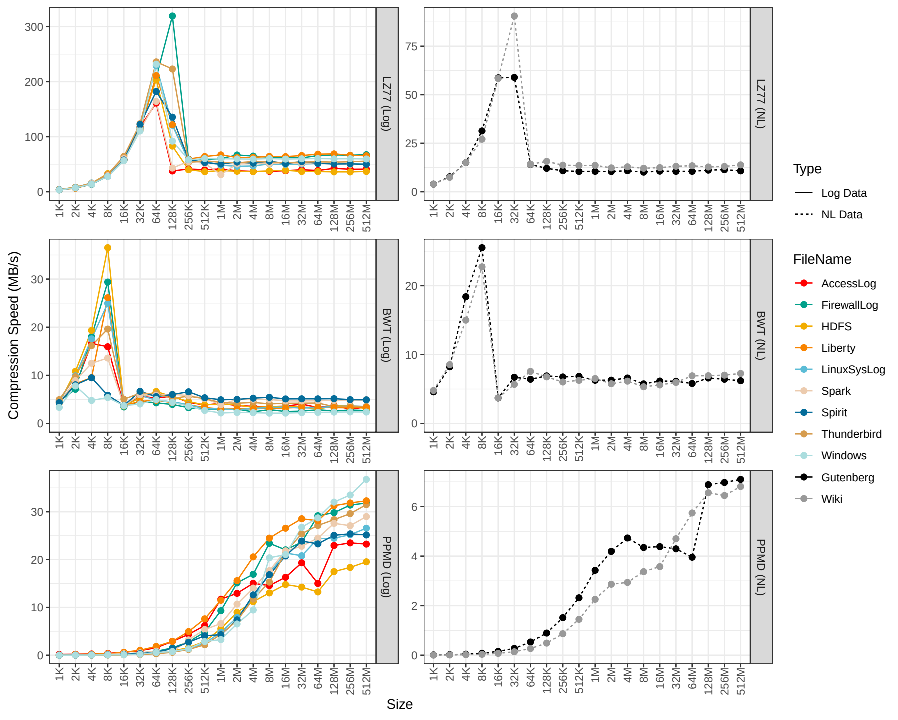

Fig. 6: Compression speed of log and natural language (NL) data at different block sizes.

log data reaches certain sizes. In particular, the compression ratio of the *LZ77* compressor at small-size (e.g., 128KB) blocks already achieves over 81.18% of the top compression ratio on log data. Having the highest compression ratio at small data sizes confirms that splitting log data into small blocks and compressing each block separately may not negatively impact the compression ratio. However, in contrast to natural language data, the compression ratio of log data is more sensitive to data sizes. For example, the compression ratio increases by 1.7 to 26.9 times when the log data size increases from 1KB to 512MB. In comparison, the compression ratio only increases by 0.6 to 1.8 times for the natural language data. Such results show that, although log data can have the best compression ratio with a small size, choosing a too small size to split log data may end up with an extremely low compression ratio. In particular, one of ELK's default options is to split log data into 16KB blocks. Figure 5 shows that the compression ratios at 16KB are only 14.63% to 80.77% of the top compression ratios.

Finding 2.2: The compression ratio of log data reaches saturation at small sizes. Practitioners need to optimize the data splitting sizes for log management tools and the rolling sizes for logging libraries.

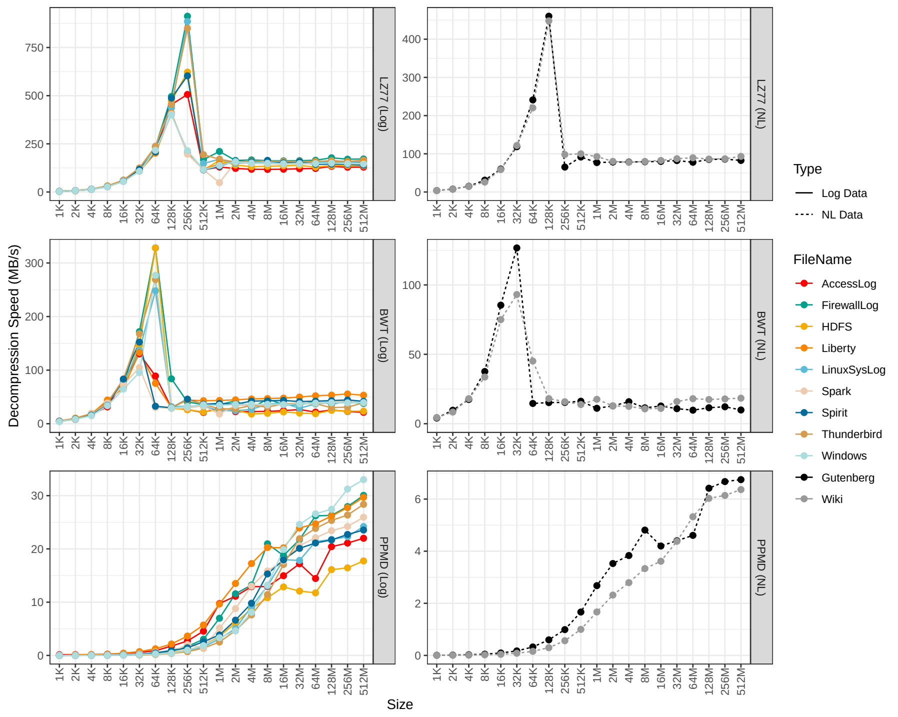

Fig. 7: Decompression speed of log and natural language (NL) data at different block sizes.

Figure 6 and Figure 7 shows the compression and decompression speeds of different compressors on the selected log and natural language data with varying sizes. The compression and decompression speeds of the *LZ77* and the *BWT* compressors increase as log size increases. The compressors reach their peak speed at small data sizes, then drops back to and stays at a lower speed as log size continues to increase. The peak compression and decompression speeds are much higher than the compression and decompression speeds at other sizes. For example, as shown in Figure 6 and Figure 7, the *LZ77* compressor reaches its peak compression speed at log sizes of 64KB or 128KB and peak decompression speed at log sizes of 128KB or 256KB for different data, respectively. In comparison, the *BWT* compressor reaches its peak compression speed at log sizes of 4KB and 8KB and peak decompression speed at log sizes of 32KB and 64KB, respectively. The compression and decompression speed of the *LZ77* and the *BWT* compressors have similar trends on the natural language data. In comparison, the *PPMD* compressor reaches its maximum compression and decompression speeds at much larger log sizes (i.e., 128MB or larger sizes).

Finding 2.3: The *LZ77* and *BWT* compressors reach peak compression and decompression speed at small log sizes (e.g., 128KB), while the *PPMD* compressor reaches its fastest compression and decompression speed on log data of large sizes.

Discussion

Dividing log data into small blocks to optimize compression performance.

In this RQ, we observe that: 1) log data is locally repetitive; 2) the compression ratio of log data reaches saturation at small log sizes; 3) for some compressors (e.g., the *LZ77* and the *BWT* compressors), the compression and decompression speeds of log data reaches its peak values at small log sizes. Therefore, depending on the selected log data, the used compressors and the specific scenarios (e.g., when compression and decompression speeds are as important as compression ratio), practitioners and log management tool providers should consider dividing log data into small blocks and compressing them separately. For example, when using the *LZ77* compressor to compress log data, it could be divided into 128KB blocks and compressed separately. Dividing log data into 128KB blocks would achieve a good tradeoff between compression ratio, compression speed, and decompression speed. When compression ratio and decompression speed are more important than compression speed, log data could be divided into 256KB blocks and compressed separately. When using the *PPMD* compressor, however, dividing log data into small blocks would not improve compression performance. As discussed in Section 2.1, Splunk divides log data into 128KB blocks and uses *gzip* (*LZ77*-based) to compress each log blocks separately. ELK divides log data into 60KB blocks (when a higher compression ratio is preferred) and uses *deflate* (*LZ77*-based) to compress each log block separately. However, none of these choices are optimal. Our findings could provide log management tool providers with evidence on optimizing the block sizes for the performance of compressors. For example, log management tools could optimize block sizes based on the characteristics of compressed log data and the usage scenarios.

The compression performance of log data often reaches saturation when log size increases to certain small data sizes (e.g., 256KB); while a too small log file may have an extremely low compression performance. Practitioners and log management tool providers should carefully choose a small data size for splitting and rolling log data.

RQ3: How do compression levels impact compression performance?

Result

Figure 8 shows the compression ratio, speed, and decompression speed of the three compressors using different compression levels. Intuitively, the compres-

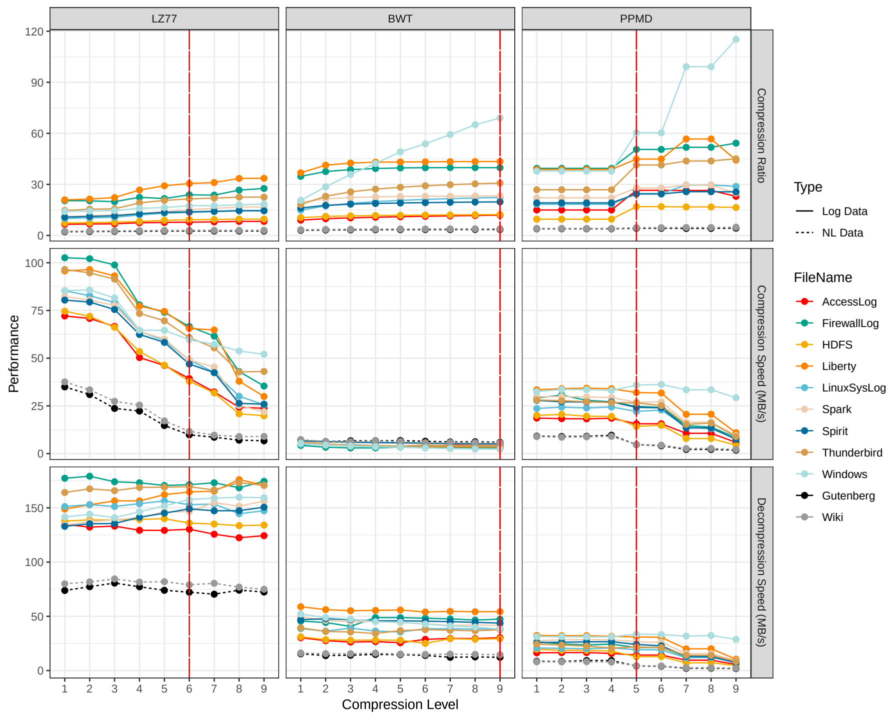

Fig. 8: Compression performance at different compression levels (red vertical lines mark the default compression levels). We used the average compression performance of ten data blocks of 512M size.

sion ratio should increase as the compression level increases, while the compression speed should decrease as the compression level increases. However, the decompression speed is impacted by the compression levels less significantly. For the *LZ77* compressor, the compression ratio and compression speed increase up to 60.67% and 276.17% on log data at different compression levels; while the decompression speed only achieves a maximum of 18.22% increase on log data among all compression levels.

Finding 3.1: The compression levels have a higher impact on compression speed and ratio than decompression speed.

Compared to the natural language data, the compression ratio of the log data is more sensitive to compression level changes; while the compression and decompression speeds are less sensitively. Hence, increasing the compression level for the log data may lead to a much higher compression ratio, while costs less compression and decompression speeds. For example, the compression ratio of the *LZ77* compressor increases 26.10% to 60.67% on the log data when comparing the highest compression ratio with the lowest compression ratio using different compression levels; however, the compression ratio

Table 4: Scott-Knott clustering on compression ratio and score under different compression levels. We use shading to highlight the top three compression levels for each file and compressor in terms of compression ratio and combined compression performance score.

Measures	Compressor	FileName	Level1	Level2	Level3	Level4	Level5	Level6	Level7	Level8	Level9
CompressionRatio	LZ77	AccessLog	9	8	7	6	5	4	3	1	2
		FirewallLog	7	8	9	5	6	3	4	2	1
		HDFS	9	8	7	6	5	4	3	2	1
		Liberty	3	3	3	2	2	1	1	1	1
		LinuxSysLog	9	8	7	6	5	4	3	2	1
		Spark	3	3	3	2	2	1	1	1	1
		Spirit	4	4	4	3	2	1	1	1	1
		Thunderbird	9	8	7	6	5	4	3	2	1
		Windows	8	7	6	5	4	3	3	2	1
		Gutenberg	9	8	7	6	5	4	3	2	1
		Wiki	9	8	7	6	5	4	3	2	1
	BWT	AccessLog	9	8	7	6	5	4	3	2	1
		FirewallLog	9	8	7	6	5	4	1	2	3
		HDFS	9	8	7	6	5	4	3	2	1
		Liberty	1	1	1	1	1	1	1	1	1
		LinuxSysLog	9	8	7	6	5	4	3	2	1
		Spark	1	1	1	1	1	1	1	1	1
		Spirit	3	2	1	1	1	1	1	1	1
		Thunderbird	9	8	7	6	5	4	3	2	1
		Gutenberg	9	8	7	6	5	4	3	2	1
		Wiki	9	8	7	6	5	4	3	2	1
	PPMD	AccessLog	4	4	4	4	1	1	2	2	3
		FirewallLog	4	4	4	4	3	3	2	2	1
		HDFS	4	4	4	4	1	1	2	2	3
		Liberty	3	3	3	3	2	2	1	1	2
		LinuxSysLog	4	4	4	4	3	3	1	1	2
		Spark	2	2	2	2	1	1	1	1	2
		Spirit	3	3	3	3	2	2	1	1	1
		Thunderbird	4	4	4	4	3	3	2	2	1
		Windows	4	4	4	4	3	3	2	2	1
		Gutenberg	4	4	4	4	2	2	3	3	1
		Wiki	4	4	4	4	3	3	2	2	1
Score	LZ77	AccessLog	5	4	3	2	1	1	1	1	1
		FirewallLog	7	7	8	5	6	3	4	2	1
		HDFS	7	6	5	4	2	1	1	3	3
		Liberty	6	5	4	3	2	1	1	1	1
		LinuxSysLog	7	6	5	4	3	2	1	1	2
		Spark	6	5	4	3	2	1	1	1	1
		Spirit	7	6	5	4	2	2	1	3	3
		Thunderbird	7	6	5	4	3	2	1	1	1
		Windows	8	7	6	5	4	3	3	2	1
		Gutenberg	5	3	2	1	2	3	4	5	5
		Wiki	5	3	2	1	1	2	3	4	4
	BWT	AccessLog	9	8	7	6	5	4	3	2	1
		FirewallLog	5	4	3	2	1	1	1	1	1
		HDFS	6	5	4	3	3	3	2	1	1
		Liberty	3	2	1	1	1	1	1	1	2
		LinuxSysLog	9	8	7	6	5	4	3	2	1
		Spark	4	3	2	1	1	1	1	1	1
		Spirit	7	6	5	4	3	2	2	1	1
		Thunderbird	9	8	7	6	5	4	3	2	1
		Windows	9	8	7	6	5	4	3	2	1
		Gutenberg	6	5	4	3	2	2	2	1	1
		Wiki	5	4	3	2	2	2	1	1	1
	PPMD	AccessLog	4	4	4	4	1	1	2	2	3
		FirewallLog	4	4	5	5	1	1	2	2	3
		HDFS	4	4	5	5	1	1	2	2	3
		Liberty	3	3	3	3	2	2	1	1	3
		LinuxSysLog	4	4	4	4	3	3	1	1	2
		Spark	2	2	2	2	1	1	1	1	2
		Spirit	4	4	4	4	1	1	2	2	3
		Thunderbird	3	3	4	4	1	1	1	1	2
		Windows	4	4	4	4	3	3	2	2	1
		Gutenberg	2	2	1	1	3	4	6	6	5
		Wiki	3	3	3	3	1	2	5	5	4

of the *LZ77* compressor on natural language data only increases 18.70% to 19.80%. The compression ratio of the *BWT* and the *PPMD* compressors on the log data can increase up to 234.93% and 204.85% using different compression levels, respectively; while the compression ratio of the *BWT* and the *PPMD* compressors only increases up to 16.39% and 17.44% on the natural language data. On the contrary, compared to natural language data, the compression speed of log data is less sensitive to compression level changes. For example, the compression speed of the *LZ77* compressor on natural language data increases 3.21 to 4.20 times from the slowest compression level to the fastest compression level. However, the compression speed of the *LZ77* compressor on log data only increases 0.65 to 2.76 times. In particular, compression performance measures may be impacted differently at certain level changes. As shown in Figure 8, the compression ratio of the *PPMD* compressor significantly increases as the compression level increases from four to five. The compression ratio of the Thunderbird log file increases 54.20% when the compression level of the *PPMD* compressor increases from four to five, with only 0.51% and 4.46% difference in compression and decompression speed.

Finding 3.2: Increasing the compression level for log data is more cost-effective (i.e., higher compression ratio with a lower cost of compression speed) than natural language data.

Table 4 shows the results of using Scott-Knott clustering to rank the compression levels of each compressor into statistically distinct groups (ranks), based on their compression ratio and combined compression performance score, respectively. In Table 4, each row represents the rank of compression ratio or the combined compression performance score for that specific compressor and file using different compression levels. For example, combined compression performance score, *LZ77*, access log, the number 1 appears five times, which means that the five corresponding compression levels are at the same rank; and the number 1 means the score is at the first rank, which is the highest rank. Practitioners could expect similar combined compression performance score using any of these five compression levels.

We observe that the highest compression level always achieves the best compression ratio for natural language data; while the highest compression level may not achieve the best compression ratio for log data. For example, the *LZ77* compressor reaches the best compression ratio on the access log file at level eight, the *BWT* compressor reaches the best compression on the Firewall log file at the level seven, and the *PPMD* compressor reaches the best compression on the HDFS log file at level five and level six. Practitioners cannot naively choose the highest compression level for the highest compression ratio for log data.

Finding 3.3: The highest compression level always achieves the best compression ratio for natural language data; while the highest compression level may not achieve the best compression ratio for log data.

Table 4 shows the results of using Scott-Knott clustering to rank the compression levels of each compressor into statistically distinct groups based on the combined compression performance scores achieved by these compression levels. For the *LZ77* compressor, the compression level seven (i.e., higher than the default compression level six) achieves the best combined compression performance scores for all log files except the Firewall log file which needs an even higher compression level (i.e., level nine) to achieve the best combined compression performance score. In comparison, the *LZ77* compressor achieves the best combined compression performance scores for both natural language data using a compression level of four (i.e., lower than the default compression level of six). For the *BWT* compressor, the default compression level (i.e., the highest level) always achieves the best combined compression performance score. The *PPMD* compressor achieves the best combined compression performance scores for log files using compression levels of five to eight, while reaching the best combined compression performance scores for natural language data using compression levels of three to five. Therefore, when compressing log data, practitioners should consider configuring compression levels that are higher than the default levels.

Finding 3.4: Log compression usually needs a compression level that is equal to or higher than the default level to achieve an optimal compression performance (based on a combined compression performance score).

Discussion

Comparing the compression ratio of the default compression level and compression levels with the highest compression ratio at different data sizes.

In RQ2, we observe that the comparison ratio may be impacted by the sizes of log data. Therefore, we studied the compression levels with the highest compression ratio with different data sizes (i.e., from 1KB to 512MB). In particular, we compared the compression ratio achieved by the default compression level with the best compression ratio achieved by a compressor. We calculated a *relative compression ratio* metric, which is the compression ratio achieved by a compression level divided by the best compression ratio achieved by all the compression levels:

$$\text{RelativeCompressionRatio}_l = \frac{\text{CompressionRatio}_l}{\text{Best compression ratio}} \tag{5}$$

For each file size, we randomly picked ten data blocks of that size and measure the average compression ratio to calculate the *relative compression ratio* metric.

Figure 9 shows the compression ratio achieved by the default compression levels relative to the best compression ratio (i.e., the relative compression ratio). For the *BWT* compressor, the default compression level is the highest, thus the default compression level always achieves the best compression ratio. For the *LZ77* compressor, the default compression level (i.e., level six) achieves a significantly smaller *relative compression ratio* for log data. For the *PPMD* compressor, the default compression level (i.e., level five) achieves nearly the best compression ratio for two of log data (i.e.., the HDFS log and the Access log data), while having significantly smaller *relative compression ratio* for the

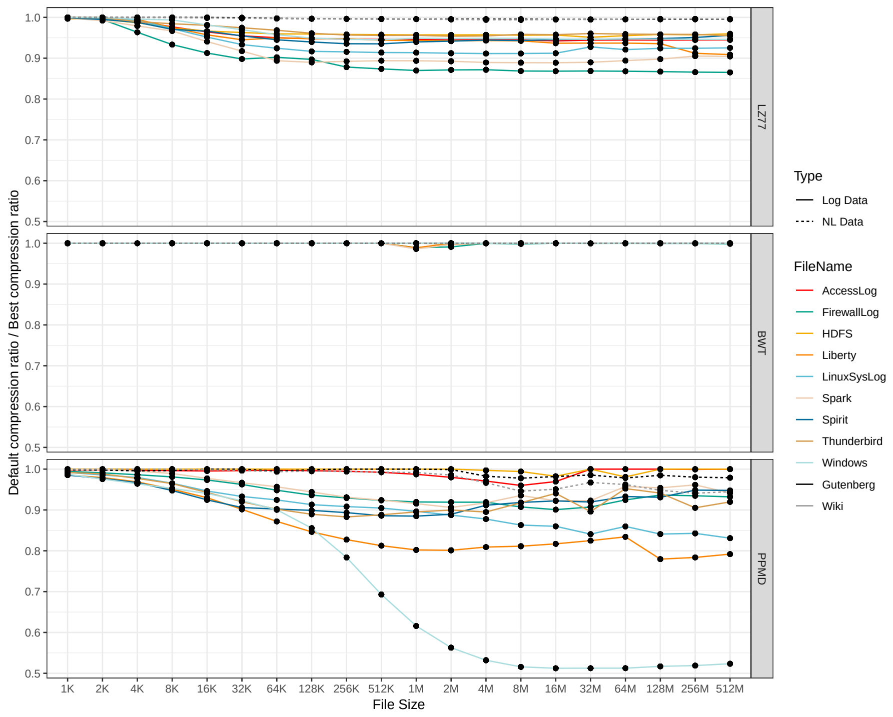

Fig. 9: Comparing the compression ratios achieved by the default compression levels to the best compression ratios.

other log data. The *relative compression ratio* of the default compression levels decreases as the size of the log file increases, and such ratio becomes stabilized at certain log sizes (e.g., around 256 KB for *LZ77*). Therefore, practitioners should consider configuring compression levels instead of simply using the default compression levels when compression log data.

As the default compression levels are usually not optimal for compressing log data, practitioners should consider configuring compression levels (in particular, higher levels) instead of simply using the default ones.

Finally, we summarize our main findings from answering our research questions in Table 5.

Table 5: The main findings of this paper.

How repetitive are log and natural language files? (Preliminary Analysis)	**Implications**
Log data is highly repetitive, showing a much lower entropy than natural language data using n-gram models.	The different repetitiveness implies a potential difference in compression performance between log data and natural language data.
How well do general compressors compress log data?(RQ1)	**Implication**
1.1 Log data is compressed faster and gets a higher compression ratio than natural language data.	None of the general compressors are optimal for all three measures. Practitioners should find the optimal compressors for their own log data (e.g., access log), considering their own usage scenarios (e.g., real-time monitoring).
1.2 The compressor with the highest compression ratio for natural language data (e.g., the *CM* compressor) is usually not the one for log data. The compressor with the highest compression ratio for one log format may not be the one for another log format either.	
1.3 Although having low compression ratios, the dictionary-based compressors achieve the fastest compression and decompression speeds for log data.	
How much does the size of a log file impact its compression performance? (RQ2)	**Implication**
2.1 Log data with a certain small size has the highest repetitiveness, i.e., log data has stronger local repetitiveness and weaker global repetitiveness.	As log data has a stronger local repetitiveness and weaker global repetitivenss, practitioners and log management tool providers should carefully choose a small data size for splitting and rolling log data.
2.2 The compression ratio of log data reaches saturation at small sizes.	
2.3 The *LZ77* and *BWT* compressors reach peak compression and decompression speed at small log sizes (e.g., 128KB), while the *PPMD* compressor reaches its fastest compression and decompression speed on log data of large sizes.	
How do compression levels impact compression performance? (RQ3)	**Implication**
3.1 Compression levels have a higher impact on compression speed and ratio than decompression speed.	As the default compression levels are usually not optimal for log compression, practitioners should consider configuring compression levels (in particular, higher levels) instead of simply using the default ones.
3.2 Increasing the compression level for log data is more cost-effective (i.e., higher compression ratio with lower cost of compression speed) than natural language data.	
3.3 The highest compression level may not achieve the best compression ratio for log data.	
3.4 Log compression usually needs a compression level that is equal to or higher than the default level to achieve an optimal compression performance (based on a combined compression performance score).	

7 Threats to Validity

7.1 External Threats

In this paper, we selected nine log data to perform our study on the performance of compressors on log data. Our empirical findings may not apply to other log data that are not covered in our study. Besides, even for log data produced by the same software system, practitioners may configure the system to output different log formats (e.g., with or without timestamps). We admit that it is difficult, if not impossible, to derive general findings that apply to all log data produced by software systems. However, based on a comprehensive literature review, our paper covers log data that is used in prior studies, followed by manual investigation and classification of log data based on their formats. The formats of our selected log data are also representative of many other formats of log data found in our investigation. We observe consistent results for our selected log data (e.g., log data is locally repetitive). Thus, our findings may apply to a broader range of log data. Practitioners may find it fruitful to examine the performance of general compressors on their own log data.

We selected 12 general compressors for our experiments. The findings derived with these compressors may not apply to other compressors. Moreover, the difference between different implementations of the same compressor may also impact our experimental results. However, in this paper, we consider the compressors that are widely adopted by prior text compression benchmarks [4, 6]. We selected 12 representative compressors across three families derived from our careful investigation. Although other compressors may have different compression performance from our selected ones, we believe our general findings (e.g., the performance of compressors on log data is sensitive to log sizes) can still apply to other compressors.

7.2 Internal Threats

In this paper, we used cross-entropy to evaluate the repetitiveness of our subject data. In particular, we used n-gram models to calculate the cross-entropy values. n-gram models are widely used to evaluate the entropy of natural language data and source code [26, 30]. However, n-gram models may not accurately capture the repetitiveness of log data. Other prediction models (e..g, DNN-based models) may achieve higher accuracy for predicting the next token in log data. However, as discussed in RQ1, the entropy values calculated using n-gram models still have a high correlation with the best compression ratios (i.e., a Spearman correlation of 0.96), indicating that n-gram model-based cross-entropy can still rank the repetitiveness of log data.

7.3 Construct Validity

In this paper, the results of compression performance are measured in our specific experimental environment. A different computing environment (e.g., different CPU speed or different implementations of some dependent libraries) may impact our experimental results. However, as we use the same experiment for all our compression experiments, we ensure that the relative compression performance measured in our compression experiments are consistent.

General compressors usually support various compression configurations (e.g., compression level, number of threads and memory limitation). In this paper, we only evaluated the impact of different compression levels on the compressors' performance. Although the compression level is not the only configuration option provided by compressors, it is usually the most important configuration option that usually impacts many other configuration options simultaneously. Future work can adopt our approaches and data to understand the impact of other configurable options on compressing log data using general compressors.

We use a state-of-the art log parsing approach [78] to extra the static and dynamic information from log messages. Such results may not reflect the absolute number of unique templates. Different regular expression statements and threshold settings could lead to varied granularity of parsing accuracy, which causes different number of templates and dynamic parameters. Although the goal of this paper is not to accurately parse log files, examining the compression results using log files that have a ground truth of static and dynamic information can further improve the validity of our findings.

7.4 Conclusion Validity

In order to understand the performance of general compressors on log files, we investigated the relationship between the characteristics (e.g., number of templates) of the studied log files and the achieved compression ratio of these log files. However, we did not observe a clear relationship between the characteristics of log files and the corresponding compression ratio. Our observation does not necessarily mean that the compression ratio is irrelevant to the characteristics of log files. Future research can explore other characteristics of log files (e.g., the similarity between different log templates) that may impact the compression ratio.

In order to take into account the random bias in measuring the compression performance on a randomly selected data block, we use the Scott-Knott test to rank the general compressors into statistically distinct groups based on their compression performance. The Scott-Knott test was designed for the division of an ANOVA experiment treatment means into homogeneous distinct groups, thus the input data of Scott-Knott is assumed to be normally distributed and have homogeneous variance [29]. Although the compression performance (i.e., the input data to the Scott-Knott test) is measured on randomly selected data blocks, we cannot guarantee that our compression performance data meets the assumptions of normality and homogeneity, which may raise a threat to the validity of our findings. Prior work [68] applies *log*-transformation on the input data of Scott-Knott to meet the assumptions. However, prior work [29] also argues that such a *log*-transformation cannot lead to the fulfillment of the assumptions.

8 Conclusions

Implementations of general compressors (e.g., *gzip*) are usually adopted to compress log data produced by software systems. However, such general compressors do not consider the characteristics of log data (e.g., produced by fixed logging statements in the source code). It is not clear how well such general compressors perform on log data. This paper performed experiments to understand the characteristics of log data, the performance of general compressors on log data, and the impact of log sizes and compression levels on the performance of compressors on log data. We observe that log data is much more repetitive than natural language data, and that the repetitiveness of log data exhibits small scopes. The compressor with the highest compression ratio for natural language data may not be the one for log data, and the compressor with the highest compression ratio for log data may not be adopted by logging libraries and log management tool providers in practice. We also observe the important role of log sizes and compression levels in the performance of general compressors on log data. In particular, general compressors achieve peak performance at small sizes of log data. Besides, the default compression level may not be optimal for compressing log data.

Our findings illustrate the challenges associated with compressing log data with general compressors, while demonstrating the opportunities for future research on customized log compression approaches. Our results can provide practitioners with insight on choosing the optimal log compressors, the sizes of log data and the levels of compressors based on their usage scenarios. Future research work on log compression can also learn from our results to improve the compression of log data by exploiting the characteristics of log data (e.g, local repetitiveness).

References

1. Siem, aiops, application management, log management, machine learning, and compliance. URL `https://www.splunk.com/`
2. 50'000€ prize for compressing human knowledge (widely known as the hutter prize) (2019). URL `http://prize.hutter1.net/`
3. Apache log4j 2 (2019). URL `https://logging.apache.org/log4j/2.x/`
4. Compression programs (2019). URL `https://maximumcompression.com/programs.php`
5. The elk stack (2019). URL `https://aws.amazon.com/elasticsearch-service/the-elk-stack/`
6. Large text compression benchmark (2019). URL `http://mattmahoney.net/dc/text.html`
7. Logback project (2019). URL `https://logback.qos.ch/`
8. Simple logging facade for java (slf4j) (2019). URL `https://www.slf4j.org/`
9. Summary of the multiple file compression benchmark tests (2019). URL `https://www.maximumcompression.com/data/summary_mf.php`
10. Wikipedia: Hutter prize (2019). URL `https://en.wikipedia.org/wiki/Hutter_Prize`

11. Aceto, G., Botta, A., Pescapé, A., Westphal, C.: Efficient storage and processing of high-volume network monitoring data. IEEE Transactions on Network and Service Management **10**(2), 162–175 (2013)
12. Augeri, C.J., Bulutoglu, D.A., Mullins, B.E., Baldwin, R.O., Baird III, L.C.: An analysis of xml compression efficiency. In: Proceedings of the 2007 workshop on Experimental computer science, p. 7. ACM (2007)
13. Awan, F.S., Mukherjee, A.: Lipt: A lossless text transform to improve compression. In: Proceedings International Conference on Information Technology: Coding and Computing, pp. 452–460. IEEE (2001)
14. Balakrishnan, R., Sahoo, R.K.: Lossless compression for large scale cluster logs. In: Proceedings 20th IEEE International Parallel & Distributed Processing Symposium, p. 435. IEEE (2006)
15. Champagne, J.: Behind the magnifying glass: How search works. `https://static.rainfocus.com/splunk/splunkconf18/sess/1523558790516001KFjM/finalPDF/Behind-The-Magnifying-Glass-1734_1538786592130001CBKR.pdf` (2018)
16. Chen, B., Jiang, Z.M.J.: Characterizing and detecting anti-patterns in the logging code. In: Proceedings of the 39th International Conference on Software Engineering, ICSE'17, pp. 71–81. IEEE Press (2017)
17. Christensen, R., Li, F.: Adaptive log compression for massive log data. In: SIGMOD Conference, pp. 1283–1284. ACM (2013)
18. Deorowicz, S., Grabowski, S.: Sub-atomic field processing for improved web log compression. In: Modern Problems of Radio Engineering, Telecommunications and Computer Science, 2008 Proceedings of International Conference on, pp. 551–556. IEEE (2008)
19. Elastic: What is the ELK stack? `https://www.elastic.co/elk-stack` (2019). (Accessed on 07/04/2019)
20. Feng, B., Wu, C., Li, J.: Mlc: An efficient multi-level log compression method for cloud backup systems. In: Trustcom/BigDataSE/ISPA, 2016 IEEE, pp. 1358–1365. IEEE (2016)
21. Fenwick, P.: Block sorting text compression. Australian Computer Science Communications **18**, 193–202 (1996)
22. Fu, Q., Lou, J.G., Wang, Y., Li, J.: Execution anomaly detection in distributed systems through unstructured log analysis. In: 2009 ninth IEEE international conference on data mining, ICDM'09, pp. 149–158. IEEE (2009)
23. Gupta, R., Gupta, R.K.: A modified efficient log file compression mechanism for digital forensic in web environment. International Journal of Computer Science and Information Technologies
24. Hassan, A., Martin, D., Flora, P., Mansfield, P., Dietz, D.: An industrial case study of customizing operational profiles using log compression. In: 2008 ACM/IEEE 30th International Conference on Software Engineering, pp. 713–723. IEEE (2008)
25. Hätönen, K., Boulicaut, J.F., Klemettinen, M., Miettinen, M., Masson, C.: Comprehensive log compression with frequent patterns. In: International Conference on Data Warehousing and Knowledge Discovery, pp. 360–370.

Springer (2003)
26. He, P., Chen, Z., He, S., Lyu, M.R.: Characterizing the natural language descriptions in software logging statements. In: Proceedings of the 33rd ACM/IEEE International Conference on Automated Software Engineering, pp. 178–189. ACM (2018)
27. He, S., Lin, Q., Lou, J.G., Zhang, H., Lyu, M.R., Zhang, D.: 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, ESEC/FSE'18, pp. 60–70. ACM (2018)
28. He, S., Zhu, J., He, P., Lyu, M.R.: Experience report: system log analysis for anomaly detection. In: 2016 IEEE 27th International Symposium on Software Reliability Engineering (ISSRE), pp. 207–218. IEEE (2016)
29. Herbold, S.: Comments on scottknottesd in response to "an empirical comparison of model validation techniques for defect prediction models". IEEE Trans. Software Eng. **43**(11), 1091–1094 (2017). DOI 10.1109/TSE.2017.2748129. URL `https://doi.org/10.1109/TSE.2017.2748129`
30. Hindle, A., Barr, E.T., Su, Z., Gabel, M., Devanbu, P.: On the naturalness of software. In: Proceedings of the 34th International Conference on Software Engineering, ICSE'12, pp. 837–847 (2012)
31. Jelihovschi, E.G., Faria, J.C., Allaman, I.B.: Scottknott: a package for performing the scott-knott clustering algorithm in r. TEMA (São Carlos) **15**(1), 3–17 (2014)
32. Jiang, Z.M., Avritzer, A., Shihab, E., Hassan, A.E., Flora, P.: An industrial case study on speeding up user acceptance testing by mining execution logs. In: 2010 Fourth International Conference on Secure Software Integration and Reliability Improvement, SSIRI'10, pp. 131–140. IEEE (2010)
33. Jiang, Z.M., Hassan, A.E., Hamann, G., Flora, P.: An automated approach for abstracting execution logs to execution events. Journal of Software Maintenance and Evolution: Research and Practice pp. 249–267 (2008)
34. Jiang, Z.M., Hassan, A.E., Hamann, G., Flora, P.: Automatic identification of load testing problems. In: Proceedings of the 2008 IEEE International Conference on Software Maintenance, ICSM'08, pp. 307–316. IEEE (2008)
35. Jurafsky, D.: Speech & language processing. Pearson Education India (2000)
36. Koehn, P.: Statistical machine translation. Cambridge University Press (2009)
37. Lemoudden, M., El Ouahidi, B.: Managing cloud-generated logs using big data technologies. In: 2015 International Conference on Wireless Networks and Mobile Communications, WINCOM'15, pp. 1–7. IEEE (2015)
38. Li, H., Chen, T.H.P., Hassan, A.E., Nasser, M., Flora, P.: Adopting autonomic computing capabilities in existing large-scale systems: An industrial experience report. In: Proceedings of the 40th International Conference on Software Engineering: Software Engineering in Practice, ICSE-SEIP'18, pp. 1–10 (2018)

39. Li, H., Shang, W., Zou, Y., Hassan, A.E.: Towards just-in-time suggestions for log changes. Empirical Software Engineering **22**(4), 1831–1865 (2017)
40. Lin, H., Zhou, J., Yao, B., Guo, M., Li, J.: Cowic: A column-wise independent compression for log stream analysis. In: Cluster, Cloud and Grid Computing, 2015 15th IEEE/ACM International Symposium on, CCGrid'15, pp. 21–30. IEEE (2015)
41. Lin, Q., Hsieh, K., Dang, Y., Zhang, H., Sui, K., Xu, Y., Lou, J.G., Li, C., Wu, Y., Yao, R., et al.: Predicting node failure in cloud service systems. In: Proceedings of the 2018 26th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering, ESEC/FSE'18, pp. 480–490. ACM (2018)
42. Lou, J.G., Fu, Q., Wang, Y., Li, J.: Mining dependency in distributed systems through unstructured logs analysis. ACM SIGOPS Operating Systems Review **44**(1), 91–96 (2010)
43. Lou, J.G., Fu, Q., Yang, S., Xu, Y., Li, J.: Mining invariants from console logs for system problem detection. In: USENIX Annual Technical Conference, pp. 1–14. ACM (2010)
44. Mahoney, M.: Large text compression benchmark. URL: http://www. mattmahoney. net/text/text. html (2011)
45. Mahoney, M.: Data compression explained. mattmahoney. net, updated May **7**, 1 (2012)
46. Mahoney, M.V.: Fast text compression with neural networks. In: FLAIRS Conference, pp. 230–234 (2000)
47. Mariani, L., Pastore, F.: Automated identification of failure causes in system logs. In: 2008 19th International Symposium on Software Reliability Engineering, ISSRE'08, pp. 117–126. IEEE (2008)
48. Mell, P., Harang, R.E.: Lightweight packing of log files for improved compression in mobile tactical networks. In: Military Communications Conference (MILCOM), 2014 IEEE, pp. 192–197. IEEE (2014)
49. Nagaraj, K., Killian, C., Neville, J.: Structured comparative analysis of systems logs to diagnose performance problems. In: Proceedings of the 9th USENIX conference on Networked Systems Design and Implementation, NSDI'17, pp. 26–26. USENIX Association (2012)
50. Navarro, G.: Compact data structures: A practical approach, chap. 11.2. Cambridge University Press (2016)
51. Oliner, A., Stearley, J.: What supercomputers say: A study of five system logs. In: 37th Annual IEEE/IFIP International Conference on Dependable Systems and Networks (DSN'07), pp. 575–584. IEEE (2007)
52. Oliner, A.J., Aiken, A., Stearley, J.: Alert detection in system logs. In: 2008 Eighth IEEE International Conference on Data Mining, pp. 959–964. IEEE (2008)
53. Otten, F., Irwin, B., Thinyane, H.: Evaluating text preprocessing to improve compression on maillogs. In: Proceedings of the 2009 Annual Research Conference of the South African Institute of Computer Scientists and Information Technologists, SAICSIT'09, pp. 44–53. ACM (2009)

54. Otten, F.J., et al.: Using semantic knowledge to improve compression on log files. Ph.D. thesis, Rhodes University (2008)
55. Pankaj Prasad and Charley Rich: Market Guide for AIOps Platforms. `https://www.gartner.com/doc/3892967/market-guide-aiops-platforms` (2018). Last accessed 04/17/2019
56. Rácz, B., Lukács, A.: High density compression of log files. In: Data Compression Conference, 2004, DCC'04, p. 557. IEEE (2004)
57. Sarbanes, P.: Sarbanes-Oxley Act of 2002. In: The Public Company Accounting Reform and Investor Protection Act (2002)
58. Sayood, K.: Introduction to data compression. Morgan Kaufmann (2017)
59. Scott, A.J., Knott, M.: A cluster analysis method for grouping means in the analysis of variance. Biometrics pp. 507–512 (1974)
60. Shannon, C.E.: A mathematical theory of communication. Bell system technical journal **27**(3), 379–423 (1948)
61. Skibiński, P., Swacha, J.: Fast and efficient log file compression. In: CEUR Workshop Proceedings of the 11th East-European Conference on Advances in Databases and Information Systems, ADBIS'07, pp. 330–342. ACM (2007)
62. Sree, P.K., Babu, I.R., et al.: Felfcnca: Fast & efficient log file compression using non linear cellular automata classifier. arXiv preprint arXiv:1312.1889 (2013)
63. Stearley, J., Oliner, A.J.: Bad words: Finding faults in spirit's syslogs. In: 2008 Eighth IEEE International Symposium on Cluster Computing and the Grid (CCGRID), pp. 765–770. IEEE (2008)
64. Syer, M.D., Jiang, Z.M., Nagappan, M., Hassan, A.E., Nasser, M., Flora, P.: Leveraging performance counters and execution logs to diagnose memory-related performance issues. In: Proceedings of the 29th IEEE International Conference on Software Maintenance, ICSM'13, pp. 110–119. IEEE (2013)
65. Tan, J., Kavulya, S., Gandhi, R., Narasimhan, P.: Visual, log-based causal tracing for performance debugging of mapreduce systems. In: 2010 IEEE 30th International Conference on Distributed Computing Systems, ICDCS'10, pp. 795–806. IEEE (2010)
66. Tan, J., Pan, X., Kavulya, S., Gandhi, R., Narasimhan, P.: Salsa: Analyzing logs as state machines. WASL **8**, 6–6 (2008)
67. Tan, J., Pan, X., Kavulya, S., Ghandi, R., Narasimhan, P.: Mochi: visual log-analysis based tools for debugging hadoop (2009)
68. Tantithamthavorn, C., McIntosh, S., Hassan, A.E., Matsumoto, K.: An empirical comparison of model validation techniques for defect prediction models. IEEE Trans. Software Eng. **43**(1), 1–18 (2017)
69. Willems, F.M., Shtarkov, Y.M., Tjalkens, T.J.: The context-tree weighting method: basic properties. IEEE Transactions on Information Theory **41**(3), 653–664 (1995)
70. Wohlin, C.: Guidelines for snowballing in systematic literature studies and a replication in software engineering. In: Proceedings of the 18th international conference on evaluation and assessment in software engineering,

p. 38. Citeseer (2014)

71. Xu, W., Huang, L., Fox, A., Patterson, D., Jordan, M.: Online system problem detection by mining patterns of console logs. In: 2009 Ninth IEEE International Conference on Data Mining, ICDM'09, pp. 588–597. IEEE (2009)
72. Yuan, D., Mai, H., Xiong, W., Tan, L., Zhou, Y., Pasupathy, S.: Sherlog: Error diagnosis by connecting clues from run-time logs. In: Proceedings of the 15th Edition of ASPLOS on Architectural Support for Programming Languages and Operating Systems, ASPLOS'10, pp. 143–154 (2010)
73. Yuan, D., Mai, H., Xiong, W., Tan, L., Zhou, Y., Pasupathy, S.: Sherlog: error diagnosis by connecting clues from run-time logs. In: ACM SIGARCH computer architecture news, vol. 38, pp. 143–154. ACM (2010)
74. Yuan, D., Park, S., Huang, P., Liu, Y., Lee, M.M., Tang, X., Zhou, Y., Savage, S.: Be conservative: Enhancing failure diagnosis with proactive logging. In: Proceedings of the 10th USENIX Conference on Operating Systems Design and Implementation, OSDI'12, pp. 293–306. USENIX Association (2012)
75. Yuan, D., Park, S., Zhou, Y.: Characterizing logging practices in open-source software. In: Proceedings of the 34th International Conference on Software Engineering, ICSE'12, pp. 102–112. IEEE Press (2012)
76. Yuan, D., Zheng, J., Park, S., Zhou, Y., Savage, S.: Improving software diagnosability via log enhancement. ACM Transactions on Computer Systems **30**(1), 4 (2012)
77. Zhu, J., He, P., Fu, Q., Zhang, H., Lyu, M.R., Zhang, D.: Learning to log: Helping developers make informed logging decisions. In: Proceedings of the 37th International Conference on Software Engineering-Volume 1, ICSE'15, pp. 415–425. IEEE Press (2015)
78. Zhu, J., He, S., Liu, J., He, P., Xie, Q., Zheng, Z., Lyu, M.R.: Tools and benchmarks for automated log parsing. In: Proceedings of the 41st International Conference on Software Engineering: Software Engineering in Practice, ICSE-SEIP'19, pp. 121–130. IEEE Press (2019)