Comparison and Model of Compression Techniques for Smart Cloud Log File Handling

Josef Spillner
Zurich University of Applied Sciences
Winterthur, Switzerland
josef.spillner@zhaw.ch

***Abstract*—Compression as data coding technique has seen approximately 70 years of research and practical innovation. Nowadays, powerful compression tools with good trade-offs exist for a range of file formats from plain text to rich multimedia. Yet in the dilemma of cloud providers to reduce log data sizes as much as possible while having to keep as much as possible around for regulatory reasons and compliance processes, many companies are looking for smarter solutions beyond brute compression. In this paper, comprehensive applied research setting around network and system logs is introduced by comparing text compression ratios and performance. The benchmark encompasses 13 tools and 30 tool-configuration-search combinations. The tool and algorithm relationships as well as benchmark results are modelled in a graph. After discussing the results, the paper reasons about limitations of individual approaches and suitable combinations of compression with smart adaptive log file handling. The adaptivity is based on the exploitation of knowledge on format-specific compression characteristics expressed in the graph, for which a proof-of-concept advisor service is provided.**

***Index Terms*—log file management, compression algorithms, text compression, benchmark, adaptivity, smart systems**

I. Introduction

Cloud computing has become a mature backbone for millions of delivered applications and services. Besides global-scale/hyper-scale infrastructure providers with dozens of data centres, many smaller managed network and platform providers are successfully covering market needs for specialised services [1]. One key issue for these providers is the handling of dynamically generated data from their services and hosted applications. Increasingly automated operations demand more insights into the provisioning and delivery situations, and therefore access to larger amounts of historic data [2]. Additionally, regulations may demand the storage of such data for longer periods of time, and occasional search for suspicious occurrences of terms. One of the most important information sources are log files, and therefore complex log management systems are set up to collect, transform and unify log messages. At the end of such pipelines, logs are compressed and stored for as long as necessary, while still being available for occasional information retrieval [3]. Consequently, providers aim at finding compression tools which squeeze the logs into the smallest possible files, while tolerating slow compression, as long as content search, in most cases preceded by decompression, should be fast. The additional cost of log management, along with monitoring and other operations, should be kept to a minimum to allow for tight pricing of offered cloud services. Increasing diversity and progress in generic compression tools and log-specific algorithms [4], [5], [6] leaves many operators without a systematic framework to choose suitable and economic log compression tools and respective configurations. This prevents a systematic solution to exploit cost tradeoffs, such as increasing investment into better compression levels while saving long-term storage cost. In this paper, such a framework is constructed by giving a comprehensive overview with benchmark results of 30 total combinations of compression tools, decompression/search tools and associated configurations.

The four concrete technical contributions of the paper are:

1) A rich graph model of compression algorithms, formats, tools, settings and runtime characteristics (`compressgraph`).
2) A robust test bench aiming at reproducible model creation with integration of relevant tools for accurate ratio and performance benchmarking (`compressbench`).
3) A reference input and results dataset of text compression and search tools applied to representative log files (`compressrefdata`).
4) A programmable advisor service that exploits the graph to recommend suitable compression for a given situation (`compressadvisor`).

All four contributions are publicly available[1]. The relation between them is summarised in Fig. 1.

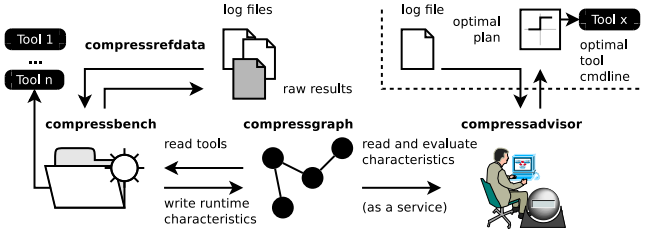

Fig. 1. Contributions of this paper

In the next sections, related works are summarised and log file scenarios defined. Afterwards, the tool comparison is introduced with the compression graph model, a testbed with curated sample data and the plan of the experiments. The results are then presented and discussed, and the advisor

[1]Contributions records: https://doi.org/10.5281/zenodo.4053735

service presented, before proceeding to an outlook on potential future compression tools that favour smart handling over the quest for raw compression ratios.

II. Related Work

In recent years, the use of online services has seen a significant growth, leading to an increase in log messages to preserve (spatial growth). For multiple reasons, including legal requirements, log files are also stored longer (temporal growth). The product of both growth factors leads to a superlinear increase in resources required to store logs. Hence, some researchers have focused specifically on new compression algorithms for log files, while others have looked into comparison approaches.

Logs can be produced by application, by system components or by network or user activities on a system. They are typically semi-structured, combining regular entry types (dates, times, hosts) with irregular user-defined messages. For a primer on application log structures and their semantic interpretation, which is also exploited by more recent compression algorithms, the work by Nimbalkar et al. [7] explains the problem domain and offers an RDF-based solution that links to domain vocabularies.

Logzip [4] has been proposed to exploit log-specific redundancy in contrast to that found in generic text. Specifically, Logzip extracts hidden structures by first sampling log lines and then clustering them by tokens and other features. One limitation of Logzip is the reliance on spaces as token separators which excludes widespread other formats. Vehicle traffic logs can be compressed semantically with high efficiency as shown in a recent study [8]. Multi-level Log Compression (MLC) [9] is another proposal aimed at compressing log files in a cloud backup workflow. It promises ratio improvements of around 16% over state of the art compression tools. Text compression beyond ASCII, applicable to the human-readable log messages, has been explored by modifications to existing byte-level compressors such as bzip2, with significant effectiveness improvements reported [10], and semantic compression for text has been investigated as well [11].

While these research prototypes are promising, a baseline comparison of widely deployed compression tools would be of immediate usefulness to operators and is in the focus of this paper. There are many benchmarks and measurements related to the comparison of compression techniques, often on specific file types. The Squash Compression Benchmark uses Squash, a generic abstraction layer running multiple codecs across configurations, files and machines, with a total of almost 60'000 individual results [12]. A general benchmark framework for compression applicable to in-memory databases has been proposed by Damme et al. [13].

In terms of benchmark results and documented comparisons, text compressions for inverted indices have been compared by Kounelis and Makris [14]. However, their work is limited to two specific techniques, OptPFD and IPC. An older comparison specifically on Java application server logs encompasses three tools, gz, bzip2 and xz [15]. There is a distinct lack of recent and comprehensive comparisons of log file compression and smart selection of best tools for this task.

A general observation can be made about the apparent business necessity of industrial compression research and tool development. This is evidenced not only by Logzip (Huawei), but also by the generic tools Brotli (Google) and Zstandard (Facebook). A second observation concerns the optimisation dimension. Most recent research works aim at a decreased compression ratio, typically at the cost of increased compression time. In contrast, another class of compression algorithms aims primarily on searchable compression with ratios being a secondary concern. Our work combines them in a common model.

III. Adaptively Compressed Log File Scenarios

Software adaptivity is controlled by goals and constraints. For compression processes, typical goals are fast compression or decompression times, fast search (often in conjunction with decompression), low-memory or low-energy (de)compression, or optimal compression ratio. The constraints are manyfold, ranging from not having the appropriate tool installed to inherent file size limitations in the tools. This knowledge needs to be captured in a knowledge base so that it can be exploited at runtime. In contrast to pure mechanical abstraction layers such as Squash, the knowledge can then lead to dynamic decisions about which codec and which parameterisation to use in any context. The novel proposal in this paper is to model the relationships in a graph, so that for instance format-equivalent compression tool alternatives can be queried dynamically based on situational context defined by goals and constraints. Through autonomous or intelligent decision-making between the possible candidates, based on an advisor service, smart adaptive log file handling is achieved.

This handling shall be illustrated by a scenario: A provider wants to store and rotate logs, asks the advisor, and gets a command-line ready to execute on the files to achieve the highest possible compression. Afterwards, the provider notices that CPU usage is high and negatively affects the business application. The constraint for less CPU involvement is brought to the advisor, leading to updated advice on a command-line that achieves still high compression with tolerable CPU load. As the higher-level choice is remembered, new tools that are added in later years are taken into account for smart and gradual self-optimisation of the system.

IV. Comparison of Compression Tools

A. Compression tools overview and graph model

The selection on compression tools is based on chronologically ordered algorithms, whereas the compression tool implementations might be newer and thus are not guaranteed to be in any temporal order. In total, 13 widely used (de)compression tools have been chosen and are modelled in the graph.

- Lempel-Ziv-Welch (LZW) (1984), implemented by (n)compress.
- Deflate (1993), implemented by zlib-flate and zip.

- Further common compression tools: gzip, bzip2, xz, zstd, LZMA/7-zip and lzop (all 1993+).
- ZPaq (2009).
- Zopfli and Brotli (2013 and 2015, respectively) [16].

Further tools, emerging from industrial and academic research, can be added to compare them on equal terms. Fig. 2 shows the base graph that relates entities among the algorithms, file formats and tools. The tools relate to command-line utilities for compression, decompression and text string search. As the graph is quite complex, the lower part of the figure zooms in on one particular entity combination with explanatory labels.

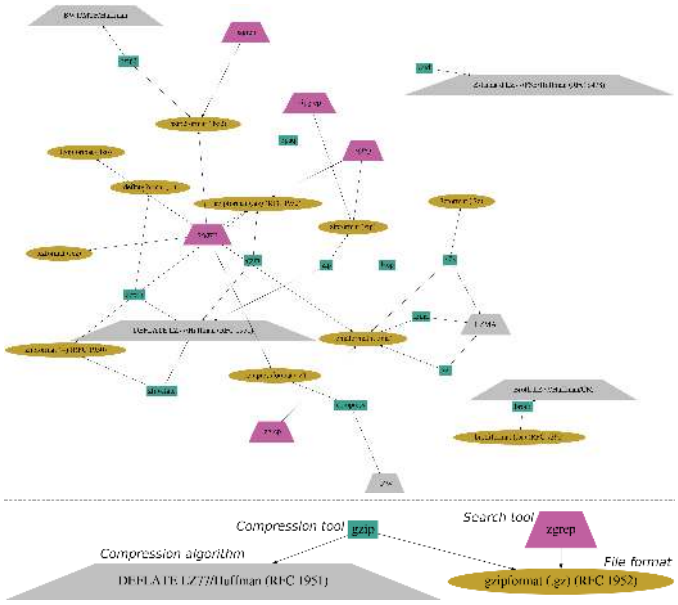

Fig. 2. above: Overview of entire base graph of compression algorithms, tools, file formats and search interfaces; below: Magnified labelled excerpt

The base graph, when shown on a large screen or printed out, is already helpful to operators to gain an overview about possible and compatible combinations. Yet considering its foreseen extensions with further tools, it might soon exceed the human cognitive ability to draw meaningful decisions based on its visual representation. Moreover, it does not yet serve as knowledge base to give answers to more complex question of high relevance in production scenarios, including the one outlined before. Such questions are for instance:

- Which tool should be used with which configuration to compress a certain file type?
- How should the tool be invoked, and how will it behave on a certain system?
- What are the implications after compression - will plain text search or regular expression search be possible?

For this reason, the base graph will later be extended into the final `compressgraph` with the benchmark results on runtime behaviour, achieved ratios and further augmented information such as direct searchability of the coded data.

B. Sample data

The Loghub [17] is a well-known reference source for heterogeneous log file formats. A final graph model is best annotated with comprehensive information about the compression-related metrics involving these files. To get a first usable graph, we use a subset of three representative server log files of different sizes and formats.

- `fw.log`: Checkpoint firewall log with approximately 20.4 million lines, 5.4 GB.
- `shp.log`: Connectivity log with around 7.3 million lines, 1.2 GB.
- `admin.log`: Firewall syslog with around 33000 lines, 4.4 MB.

C. Experiments

All experiments are conducted in a reproducible manner using `compressbench`. This benchmark tool first reads the tools to assess, along with their respective configurations like compression levels, from the base graph. It then prechecks eligibility of tools, including their installation and successful test invocation, repeats each measurement 10 times, and stores the results in structured files. Further invocation flow information about the tools is gained from the graph model, including details about suffixes created after file-based compression, whether the compressed file is deleted after decompression, and whether the (de)compression operations create files at all or use standard output that must hence be redirected.

For this paper, all experiments are conducted on a virtual machine with 2 GB RAM and without swap partitions. Although no explicit memory measurements are conducted, the compression experiments run under `oomcheck` to see if running out of memory would be an issue, which reports memory and CPU consumption as a by-product of its functionality. For the chosen tools, producing an out-of-memory error is not the case as all adapt to the available memory. It should however be stated that the different memory utilisation profiles may be another factor to consider in practice. Further, tools may crash for various reasons, including hardcoded (memory-independent) inability to handle large files.

D. Results

The experiment has been conducted in the default configuration and a single compression level per tool, except two for ZPaq due to the inherent difference of the algorithms. It ran for several days in a row due to the factors of file size, number of tool combinations, and repetitions. Fig. 3 compares the achieved compression ratios. Apart from Compress, all results stay consistently below 10% due to the high degree of redundant information in the log files. While Brotli achieves the best results, BZip2 which has been around for much longer and whose algorithm is much simpler is not off worse by much, with both achieving around 4%. ZPaq sets the optimum bar with around 2%. Apart from the ratio, auxiliary runtime behaviour is captured. For instance, Zopfli is unable to process the largest of the files, as it exceeds the 2 GB limit; hence, the

orange bar is missing for Zopfli, and the limit is accordingly recorded in the graph.

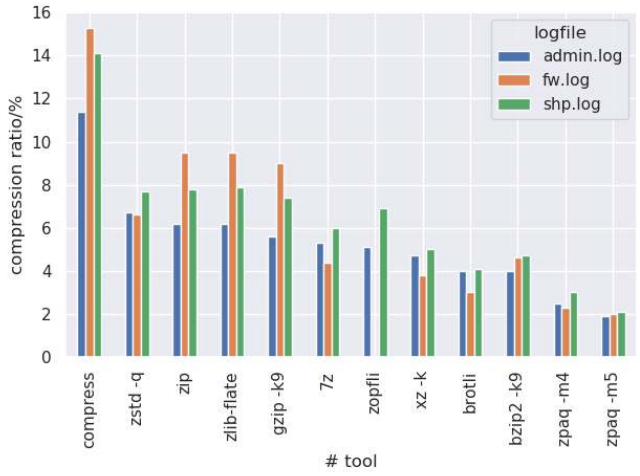

Fig. 3. Comparison of compression ratios with existing log compression tools

Fig. 4 conveys the associated compression times. It is evident that Brotli and Zopfli are least efficient. However, it should be noted that the compression time is hardly relevant in a log management scenario because most log lines are compressed incrementally as they arrive, with an arrival frequency of typically much less than 1 GB/100s most tools manage to compress. Hence, the performance interest shifts to post-compression processes such as decompression and search.

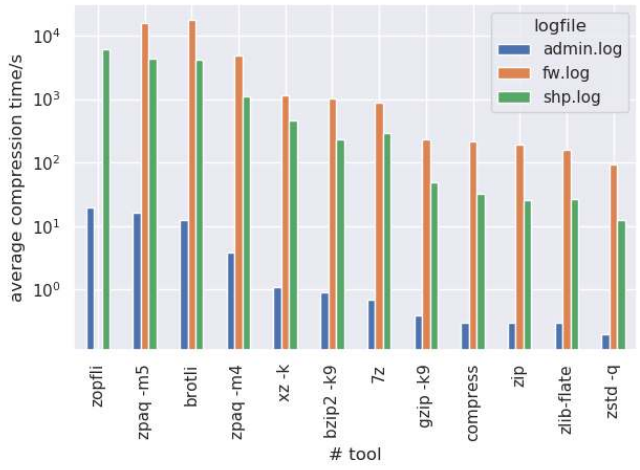

Fig. 4. Comparison of associated compression times

Fig. 5 compares the time needed to search for a substring or a regular expression. This includes the time needed to transparently decompress the file if the compression scheme does not support searchable compression. The substring searches for a concrete time stamp (`STAMP`), and the regular expression for a timestamp in conjunction with a source IP address in the firewall log file `fw.log` (`^STAMP.*src:SRC`). The search settings include both the plain and the GZip-compressed variant of the log file.

It is observable that the regular expression search (`egrep`) is faster than its substring counterpart by being able to skip whole lines when the beginning of the expression does not match. Once compression is added, this advantage is lost due to the relatively larger time component required to decompress first.

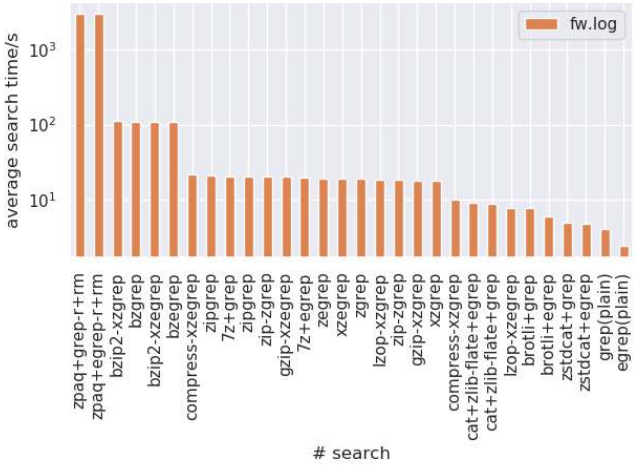

Fig. 5. Comparison of search times

The mentioned operational goal of many cloud providers is to achieve primarily low compression ratios and fast search times, with secondary concerns on compression speed or memory consumption. The correlation of compression ratios with search times is therefore a suitable guidance for choosing the most suitable tooling. Fig. 6 shows this correlation based on the experimental results. There is evidently a Pareto front; no tool is able to enter the grey triangle, while the theoretic optimum would be the lower left corner.

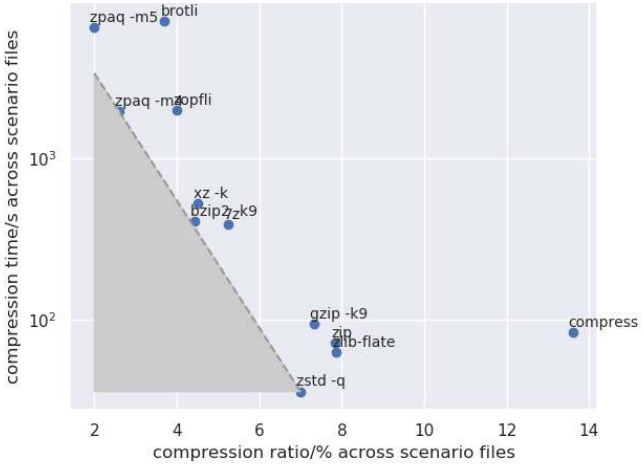

Fig. 6. Correlation of compression ratio with search times

The sample data along with the raw results is public (`compressrefdata`). Furthermore, the measured characteristics (ratios, times and resource consumption measured by `oomcheck`) are augmenting the base graph model into `compressgraph`.

V. Graph-Based Advisor Service and Adaptivity

The augmented `compressgraph` provides rich knowledge to automate smart choosing of the most suitable command-line to invoke on a given file. For instance, Zopfli and GZip are competing implementations that can process the same log file format, with just slightly better ratios and much worse compression times when using Zopfli. Moreover, Zopfli is limited by the 2 GB file barrier. In addition to this knowledge, the advisory service would therefore need to know the size (and potentially content) of the file, as well as side constraints on time and resource consumption. The advisor functionality has been implemented as a RESTful service and an interactive CLI client to be used by managed service administrators. The service narrows the possible combinations through a set of filter expressions of type `<name>::<value>`, such as `searchtool::grep`. Listing 1 shows a slightly abbreviated sample interaction with the advisor CLI in local mode, not connected to the API but sharing the same implementation.

Listing 1. Excerpt of using `compressadvisor`

```
You have 13 compression tools.
- zpaq
- zopfli
...
Your choice?
0 Quit
1 Filter on format
2 Filter on search
...
> 2
Select from a search tool below:
- zgrep
- bzgrep
...
> zgrep
Current filters:
- searchtool::zgrep
You have 5 compression tools.
...
Your choice?
...
> 1
Select from a format below:
- gzipformat
- bzip2format
...
> gzipformat
Current filters:
- searchtool::zgrep,format::gzipformat
You have 4 compression tools.
- zpaq
- gzip
- zstd
- lzop
Your choice?
...
```

Beyond search tool and file format, the advisor accepts the following filters:

- `filesize`, to indicate the size of the file to be (de)compressed.
- `maxmem`, maximum memory consumption during (de)compression.
- `priority`, an ordered list of ranking criteria encompassing `compresstime`, `compressratio` and `decompresstime`.
- `searchable`, boolean flag to exclude schemes without direct search support

In addition to the filters, which follow a typical filter syntax of `key::value`, the RESTful API also takes regular parameters with a syntax of `key=value`. The parameters are:

- `filename`, template placeholder to place the filename into the right position in the produced system command. This parameter is required.
- `verbose`, to produce rich log output on the process of command-line construction which is then written to the web server log.

The service interface is aimed at automated systems operations and directly delivers the command to execute. Obviously, along with the potential use of any public data or API this raises security concerns, which are not addressed in this paper as the complexity of holistic system security is high by itself. It is assumed, and acknowledged as limitation of the work, that the advisor should only be run in trusted environments with a protection of the graph against unauthorised modifications. To make users at least aware of potential security problems, two control points are defined for consumers of the API including the CLI client: The first allows for inspecting the chosen command to execute. The second records previous invocations and allows to define a notification hook for reporting any deviations.

Listing 2 shows how to run a compression using the advisor service using both the CLI wrapper in non-interactive mode, connected to the API, and the raw HTTP access to the API itself. In contrast to the interactive mode of the CLI, the response of the API and its CLI wrapper is always guaranteed to include one result. If multiple results are found after filtering, the first one will be returned consistently. If no results are found, the returned command refers to the advisor CLI itself which outputs an appropriate error message so that the compression will not fail silently.

Listing 2. RESTful query of `compressadvisor`

```
# CLI API wrapper, determines file size
    automatically
compressadvisor compress prod.log format::gzip
# API
$(curl -sSX GET "http://compressadvisor/compress?
    format::gzip&filesize::2gb&filename=prod.log")
```

To elaborate on the integration options, Fig. 7 shows the possible workflows. In security-sensitive environments, the benchmark would be first re-run to build a trustworthy private graph, and all commands determined by it would be verified manually before put into production along with notifications in case the command differs from previous invocations. For

convenience in non-critical environments, a publicly hosted API could be used directly. In order to demonstrate the convenience, the advisor service has been deployed to the Google Cloud Platform and is publicly available for interested users[2].

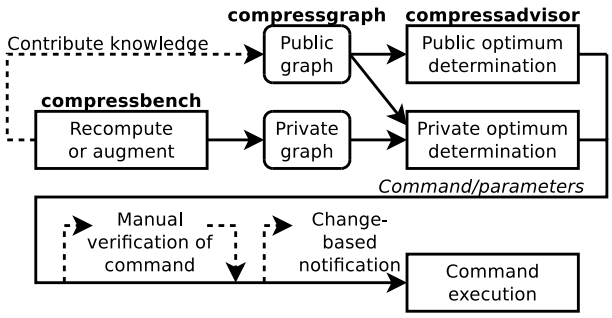

Fig. 7. Flexible workflows around the `compressadvisor` service

VI. Conclusions

In the recent decade, several new compression tools were introduced and others were updated. Further research on better compression techniques continues, mainly driven by commercial interests of global players in need of handling huge amounts of data in a cost-efficient way. With the comparison presented in this paper, operators of managed services, who are among the key users of balanced compression needs, first get an up-to-date overview about the behaviour of currently available tools. They can further keep it up to date by repeating the experiments and extending the covered tools as needed. Furthermore, they can exploit the compression-related characteristics graph as knowledge base to control adaptive compression based on system constraints and differing needs concerning future search through the compressed files. We expect smart and knowledge-based compression to extend into other areas such as efficient caching [18]. Moreover, as mentioned in the introduction, the larger problem extends to the economics of data handling, thus we expect advanced knowledge graphs that take per-provider compute and storage cost into account.

Acknowledgments

This material is based upon work supported by Google Cloud, with GCP Research Credits for Serverless Data Integration. Research partially supported by Swiss Leading House for the Middle East and North Africa, with funds of the Swiss State Secretariat for Education, Research and Innovation (SERI) under grant Application of stealth computing in highly information-sensitive cloud environments.

References

[1] Antaviana. CloudTango – World's largest MSP directory. online: https://www.cloudtango.org/, August 2020.

[2] Advisor online: https://compressadvisor.oa.r.appspot.com

[2] Bartosz Balis, Michal Orzechowski, Krystian Pawlik, Maciej Pawlik, and Maciej Malawski. Cloud infrastructure automation for scientific workflows. In Roman Wyrzykowski, Ewa Deelman, Jack J. Dongarra, and Konrad Karczewski, editors, *Parallel Processing and Applied Mathematics - 13th International Conference, PPAM 2019, Bialystok, Poland, September 8-11, 2019, Revised Selected Papers, Part I*, volume 12043 of *Lecture Notes in Computer Science*, pages 287–297. Springer, 2019.

[3] Chao-Tung Yang, Endah Kristiani, Yuan-Ting Wang, Geyong Min, Ching-Han Lai, and Wei-Je Jiang. On construction of a network log management system using ELK stack with ceph. *J. Supercomput.*, 76(8):6344–6360, 2020.

[4] Jinyang Liu, Jieming Zhu, Shilin He, Pinjia He, Zibin Zheng, and Michael R. Lyu. Logzip: Extracting hidden structures via iterative clustering for log compression. In *34th IEEE/ACM International Conference on Automated Software Engineering, ASE 2019, San Diego, CA, USA, November 11-15, 2019*, pages 863–873. IEEE, 2019.

[5] Balázs Rácz and András Lukács. High density compression of log files. In *2004 Data Compression Conference (DCC 2004), 23-25 March 2004, Snowbird, UT, USA*, page 557. IEEE Computer Society, 2004.

[6] M. Kösters, J. Leufken, Stefan Schulze, K. Sugimoto, Joshua Klein, R. P. Zahedi, Michael Hippler, S. A Leidel, and Christian Fufezan. pymzml v2.0: introducing a highly compressed and seekable gzip format. *Bioinform.*, 34(14):2513–2514, 2018.

[7] Piyush Nimbalkar, Varish Mulwad, Nikhil Puranik, Anupam Joshi, and Tim Finin. Semantic interpretation of structured log files. In *17th IEEE International Conference on Information Reuse and Integration, IRI 2016, Pittsburgh, PA, USA, July 28-30, 2016*, pages 549–555. IEEE Computer Society, 2016.

[8] András Gazdag, Levente Buttyán, and Zsolt Szalay. Efficient lossless compression of CAN traffic logs. In Dinko Begusic, Nikola Rozic, Josko Radic, and Matko Saric, editors, *25th International Conference on Software, Telecommunications and Computer Networks, SoftCOM 2017, Split, Croatia, September 21-23, 2017*, pages 1–6. IEEE, 2017.

[9] Bo Feng, Chentao Wu, and Jie Li. MLC: an efficient multi-level log compression method for cloud backup systems. In *2016 IEEE Trustcom/BigDataSE/ISPA, Tianjin, China, August 23-26, 2016*, pages 1358–1365. IEEE, 2016.

[10] Adam Gleave and Christian Steinruecken. Making compression algorithms for unicode text. In Ali Bilgin, Michael W. Marcellin, Joan Serra-Sagristà, and James A. Storer, editors, *2017 Data Compression Conference, DCC 2017, Snowbird, UT, USA, April 4-7, 2017*, page 441. IEEE, 2017.

[11] Dariusz Ceglarek. Semantic compression for text document processing. *Trans. Comput. Collect. Intell.*, 14:20–48, 2014.

[12] Evan Nemerson. Squash Compression Benchmark. online: https://quixdb.github.io/squash-benchmark/, December 2018.

[13] Patrick Damme, Dirk Habich, and Wolfgang Lehner. A benchmark framework for data compression techniques. In Raghunath Nambiar and Meikel Poess, editors, *Performance Evaluation and Benchmarking: Traditional to Big Data to Internet of Things - 7th TPC Technology Conference, TPCTC 2015, Kohala Coast, HI, USA, August 31 - September 4, 2015. Revised Selected Papers*, volume 9508 of *Lecture Notes in Computer Science*, pages 77–93. Springer, 2015.

[14] Fotios Kounelis and Christos Makris. Comparison between text compression algorithms in biological sequences. *Inf. Comput.*, 270, 2020.

[15] Stefan Seidel. Log file compression. online: https://www.stefanseidel.info/Log_file_compression, June 2012.

[16] Jyrki Alakuijala, Andrea Farruggia, Paolo Ferragina, Eugene Kliuchnikov, Robert Obryk, Zoltan Szabadka, and Lode Vandevenne. Brotli: A general-purpose data compressor. *ACM Trans. Inf. Syst.*, 37(1):4:1–4:30, 2019.

[17] Jieming Zhu, Shilin He, Jinyang Liu, Pinjia He, Qi Xie, Zibin Zheng, and Michael R. Lyu. Tools and Benchmarks for Automated Log Parsing. In *International Conference on Software Engineering (ICSE)*, 2019.

[18] Biswabandan Panda and André Seznec. Dictionary sharing: An efficient cache compression scheme for compressed caches. In *49th Annual IEEE/ACM International Symposium on Microarchitecture, MICRO 2016, Taipei, Taiwan, October 15-19, 2016*, pages 1:1–1:12. IEEE Computer Society, 2016.