
Production Systems and Information Engineering
Volume 11 (1), pp. 85–101.
https://doi.org/10.32968/psaie.2023.1.7

EFFICIENCY AND PERFORMANCE EVALUATION OF OPEN-SOURCE BACKUP TOOLS – AN EMPIRICAL ANALYSIS OF STORAGE FOOTPRINT AND EXECUTION TIME

ÁRON KISS
University of Miskolc
Hungary Institute of Information Technology
aron.kiss@uni-miskolc.hu

KÁROLY NEHÉZ
University of Miskolc
Hungary Institute of Information Technology
karoly.nehez@uni-miskolc.hu

Abstract. Information is one of the most valuable assets in the digital age, and regular, reliable backups are essential to protect against hardware failures, malicious attackers, and human mistakes. While deduplication and compression have well-known advantages in data storage, a wide variety of open-source backup solutions available in the market raises the question: do these similar tools differ significantly in terms of backup time and resource efficiency? This article aims to answer that question by conducting experiments with diverse datasets and evaluating the performance of 4 selected open-source backup tools by examining runtime and storage requirements. Our analysis of measurements provides insights to assist potential users in making informed decisions about their backup architecture.

Keywords: information security, backup software, data deduplication, software evaluation, open-source software

1. Introduction

In the digital age, data is one of the most valuable assets for individuals and organizations. With increasing volume and complexity of data, ensuring its protection has become a primary concern. Among the myriad of strategies to secure data, regular backups are standing as a fundamental pillar in the realm of data protection. Backup solutions play a crucial role in preserving the integrity, availability, and recoverability of data, serving as a safety net against many threats, from hardware failures to cyberattacks and human mistakes [1, 2].

Efficient and reliable backup practices are a necessity in our data-driven world. As digital services continue to expand, the storage of redundant or obsolete data poses a substantial burden in terms of both storage capacity and costs. This is where deduplication and compression technologies come into play, offering a promising avenue to mitigate the storage footprint of backed-up data and economize the associated costs. By eliminating redundancy and reducing the size of backups, deduplication and compression optimize storage resources and deliver significant economic benefits to organizations [3].

While the benefits of deduplication and compression are well recognized, the landscape of open-source backup tools presents a broad and diverse array of options [4]. As open-source solutions gain popularity for their cost-effectiveness, flexibility, and transparency, organizations and individuals are faced with many choices. However, this diversity raises a critical question:

"Do backup software with a similar purpose and architecture show significant deviation in runtime or compression efficiency?"

This article addresses this pivotal question by conducting an empirical analysis of a subset of open-source backup tools. We will evaluate the efficiency and performance of various backup solutions, shedding light on their storage footprint and execution time. Through this study, we pursue to assist users in making decisions that align with their specific data protection needs.

In the following sections, we present the methodology, the experimental setup, and the results of our empirical analysis, providing insights into open-source backup tools' landscape and diverse capabilities in effectively safeguarding and managing data.

2. General steps of the backup process

In this chapter, we provide an overview of the general architecture of modern software solutions designed for creating deduplicated backups, along with a presentation of the most common steps involved in the backup creation process.

This process involves data deduplication, compression, and encryption. After that, the data is uploaded to a repository with an arbitrary storage architecture.

2.1. Data deduplication

Deduplication is a method aimed at removing redundant data from storage, comprising three distinct subprocesses: *chunking*, *fingerprinting*, and *indexing*:

- *Chunking* is the method of segmentation of information into discrete pieces of data.
- *Fingerprinting* (or *hashing*) entails verifying whether a chunk is redundant; if it's new, the chunk is uploaded along with its fingerprint to the repository,

whereas if it's redundant, only the fingerprint is uploaded.

- *Indexing* revolves around the management and upkeep of fingerprints associated with existing chunks.

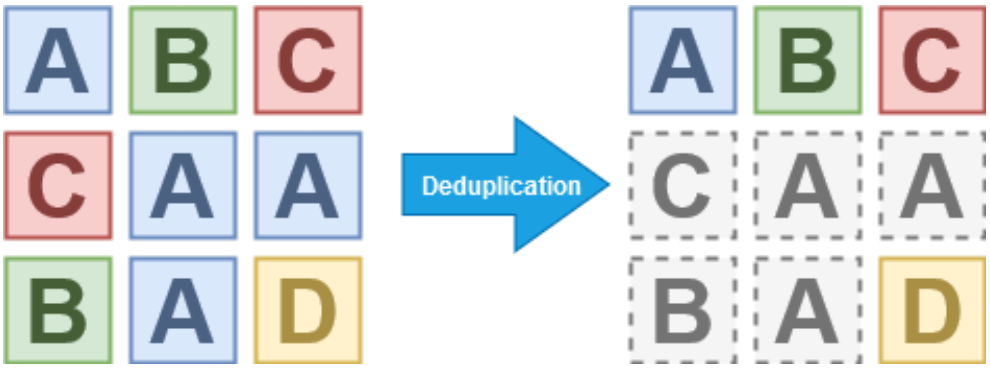

Figure 1. Redundancy reduction by deduplication

Figure 1 shows the general idea of data deduplication. The colored squares on the left represent chunks of the data to be backed up. When the data is redundant, it can contain multiple chunks that store the same information. When a deduplicating backup is made on this dataset, unique chunks must be saved. Chunks with dashed line borders are only references to already saved chunks. Along this line of thinking, a coarse-grained dataset compression can be reached.

Deduplication can occur either at a file or block level. File-level deduplication operates by identifying and removing duplicate files, while block-level deduplication operates at a more granular level, where it can eliminate duplicate blocks, which can be either of a fixed or variable size. File-level deduplication offers the advantage of being resource-efficient, however, its limitation lies in its inability to eliminate smaller redundant data chunks that are smaller than an entire file. Block-level deduplication excels at eliminating smaller data chunks, surpassing the capabilities of file-level deduplication in this regard. Nevertheless, its drawback is its higher resource demands.

Modern, general-purpose backup software typically provides block-level deduplication by employing fixed-size or content-defined chunking algorithms.

2.1.1. Fixed-Size Chunking (FSC)

The data stream is divided into equal-sized pieces in the case of fixed-size chunking. One significant advantage of this method is its low computational demand. However, fixed-size chunking generally tends to achieve a lower deduplication ratio than variable-sized methods.

The reason behind this is the *boundary-shift problem*, which causes the phenomenon, that even the most negligible modification in a file leads to entirely new chunks [6].

An example of this can be seen in Figure 2., the bits in the first row represent the structure of the original file. The bits in the second row match those of the original file, except that a new bit has been inserted at the beginning. This small change of only one bit results in creating entirely new chunks that are then saved to the backup repository.

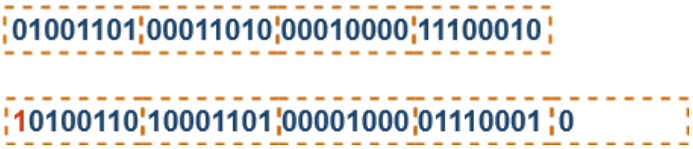

Figure 2. The boundary-shift problem

2.1.2. Content-Defined Chunking (CDC)

In the case of Content-Defined Chunking, the data stream is not divided into fixed-size portions but rather, the chunk sizes vary based on the content, only the maximal chunk size can be defined. This approach requires more computational resources than the fixed-size method, but it is not sensitive to the boundary-shift problem and produces a higher deduplication ratio [5].

Most CDC algorithms process the data stream in a sliding window, generating rolling hash values. Rabin's fingerprint is one of the oldest and most efficient hash functions, however it has a high computational overhead [9, 10].

An increasing number of optimized CDC algorithms are being published. [12] proposes a novel method called Asymmetric Extremum (AE), which offers speed advantages compared to traditional chunking solutions. [11] introduces an innovative approach faster than AE, while achieving a deduplication ratio that approaches or surpasses Rabin-based CDC. [7] proposes a cosine similarity-based fuzzy interference system to identify similar chunks.

2.1.3. Fingerprinting

During fingerprinting, a hash value should be generated for each specified chunk. This value uniquely identifies the content of the chunk. This is typically done using collision-resistant cryptographic hash functions (e.g., SHA-256). These generated fingerprints are then stored in a database and used for identifying duplicate chunks.

2.2. Compression

While deduplication does an efficient, coarse-grained compression of the data stream, redundancy may occur in the content of the unique chunks generated as the output of the deduplication process. Conventional compression algorithms are used to carry out the fine-grained compression of these chunks.

Dictionary-based compression algorithms are commonly built upon the LZ77 and LZ78 algorithms. In modern backup tools, variants of these algorithms are

employed, such as DEFLATE and LZMA, which are focused on compression ratios, while LZO and LZW algorithms offer significant improvements in compression speed [14].

Novel algorithms are also present in backup products. Zstandard developed in 2015 by Yann Collet was created to achieve compression ratios like the DEFLATE algorithm but with a focus on speed, particularly during decompression [13].

2.3. Encryption

By leveraging cloud-based storage capabilities, organizations and individuals are confronted with security issues related to their data. One such problem is the lack of transparency in infrastructure operations. While a breach of such an incident would undoubtedly lead to a negative business impact for the operator, there is usually no guarantee that administrators cannot access the content of the storage.

Another significant problem is that multiple tenants share a typical cloud-based storage service. Since users do not have precise insights into the circumstances of infrastructure management, they cannot be entirely certain that the isolation between tenants is adequate, and that malicious intruders cannot compromise it.

To address these security concerns, most backup tools incorporate built-in encryption solutions, often offering end-to-end encryption and data obfuscation to prevent fingerprinting attacks. The most frequently applied encryption methods are AES256 and GPG.

AES256 is a symmetric-key encryption algorithm and is considered highly secure due to its robust encryption key size, making it computationally infeasible by brute-forcing.

GPG is an open-source encryption software that utilizes a combination of asymmetric and symmetric encryption techniques, allowing users to encrypt data using the recipient’s public key, which can only be decrypted with the recipient's corresponding private key. GPG provides a robust and trusted method for securing sensitive information and ensuring data integrity.

2.4. Storage

Modern backup software stores deduplicated chunks in repositories. Ensuring rapid access to these chunks is crucial during the backup and restore processes. This is achieved through various indexing solutions [14].

The schematic structure of such a repository is depicted in Figure 3.

The repository includes a *chunk storage* component for storing individual chunks and their corresponding fingerprints.

The repository also contains containers for *restore points* (often called *snapshots*, *archives* or *versions* in actual software). Each restore point captures the current state of the file system at the moment of the backup. Restore points store not only the

paths of the saved files but also their actual content, referencing the identifiers of chunks stored in the chunk storage.

As a result of this storage logic, the repository provides "synthetic full backups", instead of incremental or differential backups to the users. This means that each restore point serves the entire state of the backup target directory, but identical chunks are not stored multiple times for each restore point.

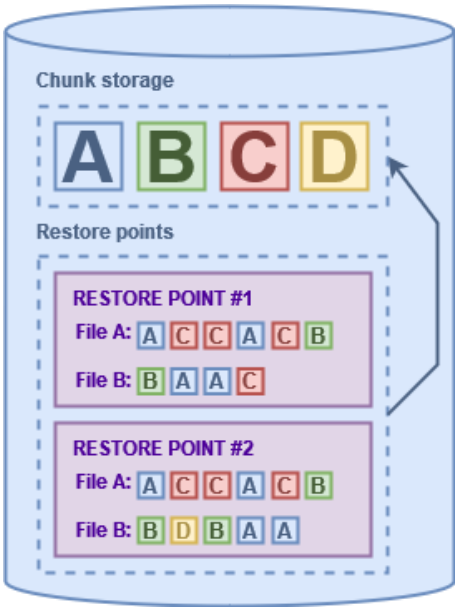

Figure 3. A general backup repository architecture

The underlying storage architecture of the backup repository is very diverse. Most backup tools support not only traditional storage locations like local disks and network-attached storage, but also remote storage accessible through standard protocols (e.g., SSH, FTP, WebDAV) and provider-specific "Platform as a Service" solutions (e.g., Amazon S3, Google Cloud Storage, Azure Blob Storage).

A widely used solution to achieve this cross-compatibility is Rclone, a software designed for file management in the cloud, offering a robust alternative to the web storage interfaces provided by various cloud vendors. It supports over 70 cloud storage services, business and consumer file storage solutions, and standard transfer protocols. [15]

3. Overview of the examined software

To examine the current backup landscape, we selected four popular tools that exhibit partial differences in their characteristics and features. The most essential information regarding these tools can be found in Table 1.

Table 1
Main characteristics of the examined backup programs

	borg	**duplicati**	**duplicity**	**restic**
First release	11-06-2015	01-06-2008	26-08-2002	14-09-2015
Last release	24-03-2023	25-05-2023	27-09-2023	31-07-2023
Git starrers	9.817	9.318	198	21.294
Language	Python, C	C#	Python, C	Go
Platform	Linux Mac	Windows Linux Mac	Linux Mac	Windows Linux Mac
Repository backends	Local SSH	Local SSH FTP, SFTP S3 Azure B. S. Google C. S. Swift WebDAV SharePoint rclone etc.	Local SSH FTP, SFTP S3 Azure B. S. Google C. S. Swift WebDAV SharePoint rclone etc.	Local SFTP S3 Azure B. S. Google C. S. Swift rclone etc.
Chunking ([d] = default)	CDC[d] FSC	FSC[d]	CDC[d]	CDC[d]
Compression ([d] = default)	LZ4[d] ZStandard zlib LZMA	DEFLATE[d] LZMA2	GZip[d]	ZStandard[d]
Encryption ([d] = default)	AES-256[d]	AES-256[d] GPG	GPG[d]	AES-256[d]
Built-in scheduling	No	Yes	No	No
User interface	CLI	CLI + GUI	CLI	CLI
License	BSD 3-Clause	GNU LGPL	GNU GPL	BSD 2-Clause

We selected two mature and two relatively recent backup software solutions for the study. *duplicati* and *duplicity* have been present in the backup product market for over a decade, *borg* and *restic*, while having less extensive histories, have both garnered large user bases.

borg (formerly *attic*) was initiated in 2015. It is called a deduplicating backup program, which optionally supports compression and authenticated encryption [22]. One of *borg*'s advantages over its competitors is its extensive customization options, stemming from the ability to choose the chunking algorithm and compression method. However, it only supports local disk and SSH as backends to store its repository.

duplicati is a mature cross-platform backup solution that offers a high level of user-friendliness with GUI support and built-in scheduling functionality [23]. In contrast to its competitors, it employs fixed-size, rather than content-defined chunking during data deduplication [24].

duplicity is the oldest among the examined backup tools. It secures directories by creating encrypted tar-format volumes, which are then uploaded to a remote or local file server. It is a part of the Fedora, Debian, and Ubuntu distributions of GNU/Linux [25].

restic is a cross-platform backup program that is designed to be "fast, efficient and secure". It is a solution that prioritizes simplicity, ensuring that setting up and restoring backups is effortless. It strongly emphasizes security, using encryption to safeguard data [21]. It has the most significant number of followers among the examined software, and also proved its robustness in "CERNBox", the cloud collaboration hub at CERN with more than 37,000 user accounts [20].

4. Design of the experiments

In this chapter, we introduce the experimental setup and methodology used to perform measurements and describe the properties in detail of the data sets used to run experiments.

4.1. Experiment methodology

We executed separate initial, "complete" backups of the datasets using the software tools under examination. For these evaluations, we utilized the most recent software versions available as of the date specified in the "Last release" column of Table 1. During the evaluation process, we focused on measuring two primary factors: the total execution time and the efficiency of deduplication and compression. We refer to this efficiency as the "optimization ratio", and calculated it as follows:

$$Optimization\ ratio = \frac{Original\ data\ size}{Deduplicated\ and\ compressed\ data\ size}$$

For the test runs, we configured a maximum chunk size of 4 MB. For each execution, we applied the examined software's default chunking algorithm, encryption method, and compression technique. The backups were created to the local file system.

4.2. Testbed

The experiments are executed in a VM with the following parameters:

CPU:	Intel Core i7-12700H 14 cores, 14 threads
RAM:	16 GB
Storage:	Kingston SNV2S/1000G SSD 80 GB VDI virtual disk image, ext4 filesystem
OS:	Ubuntu 22.04.3 LTS

To ensure the reliability of our results, we repeated these measurements three times for each combination of dataset and software. Our analysis will rely on the averages derived from these measurements as the basis for our assessment.

4.3. Document archive (DS1)

Our first dataset was a document archive. In archives of this kind, one can typically find a substantial volume of files, which tend to be relatively large in size (compared to Linux system binaries), and many of them may already be compressed to some extent.

These archives' size results from the sheer quantity of content and the inclusion of high-resolution multimedia elements. Due to the prevalence of compression techniques, such as LZW and DEFLATE algorithms, which are used in the popular PDF and DOCX file formats, these files are stored in relatively space-efficient before the deduplication [16, 17].

As a basis of this dataset, we downloaded all the issues of *Magyar Közlöny* and its appendix *Hivatalos Értesítő* published between 01-01-2013 and 14-09-2023. The dataset contains 3141 PDF files, the overall size of the dataset is approx. 4178 MB. The distribution of file sizes are presented in Figure 4. Average file size is 1.33 MB, the median size is 0.73 MB and the standard deviation is 2.88 MB.

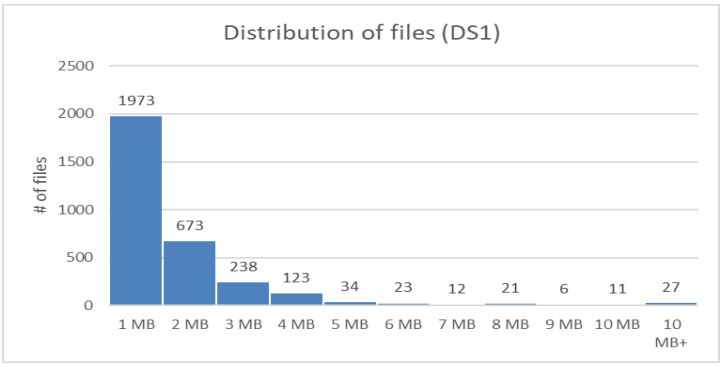

Figure 4. Distribution of file sizes in DS1

4.4. Clean server environment (DS2)

Our second dataset contains several clean system environments with files that define the initial state of the system for running specific applications. These files are instrumental in establishing the baseline configurations required for the smooth operation of individual applications. In essence, they encapsulate the essential settings, binary dependencies, source codes and other prerequisites necessary to initialize and run these applications effectively within a given system environment.

We collected data for this dataset by installing a fresh Docker daemon on a machine and pulling 30 of the most popular Docker images to the system. Following that, we backed up all files belonging to the Docker ecosystem, including all the contents of /var/lib/docker/.

The distribution of files based on their formats is illustrated in Table 2. The examined dataset contains an array of uncompressed textual files (including PHP, Python, JavaScript code files, JSON data, and plain text documents), offering many possibilities for redundancy reduction [18, 19]. However, in addition to these predominantly uncompressed formats, the dataset also features a presence of less redundant, compressed file formats such as PNG and GZ. This diverse mix of file types indicates higher redundancy than DS1.

Table 2
Distribution of files by format in DS2

# of files	File format	# of files	File format
26443	.php	4151	.pm
18837	.h	3795	.so
16880	.py	2973	.txt
11063	.js	2535	.mo
9398	.go	1787	.css
9386	.pyc	1737	.beam
9337	.json	1595	.png
5383	.pl	34145	other files with extension
5300	.gz	33972	other files without extension

The distribution of file sizes is presented in Figure 5. This dataset contains 198,717 files, the overall size of the dataset is approx. 9,761 MB. The average file size is 49.12 KB, the median size is 2.47 KB, and the standard deviation is 1.13 MB.

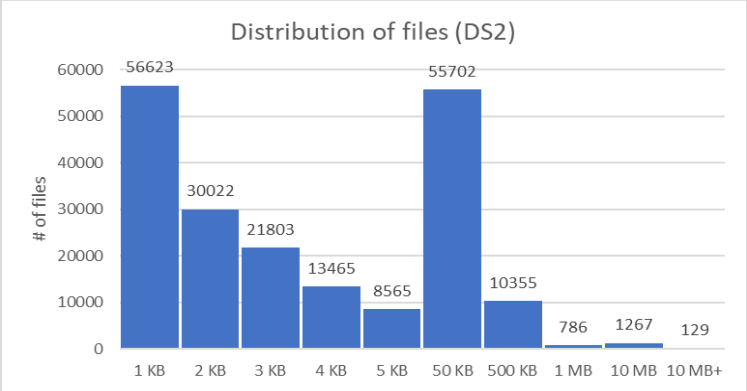

Figure 5. Distribution of file sizes in DS2

4.5. Live server environment (DS3)

Our third dataset is a replication of a real server environment. It runs on Ubuntu operating system, and several JavaScript-based applications are in use.

A significant distinction from DS2 is that this dataset contains not only the binaries, source code, and initial configuration of the running applications but also a wide array of user data, log files, caches, and databases.

That dataset is a snapshot of a server's activity and interactions beyond just the core application components. This additional data differentiates from DS2, indicating a higher level of redundancy.

It can be seen in Table 3, that this sample predominantly contains files in uncompressed textual formats. These include source code files, logs, documentation files, vector graphic images, configuration files, etc.

Table 3
Distribution of files by format in DS3

# of files	File format	# of files	File format
91508	.js	2993	.h
37801	.log	2311	.py
27542	.ts	2116	.pyc
15008	.map	1778	.txt
11060	.pem	1655	.rst
10406	.json	1368	.vim
10198	.md	1286	.yml
4799	.gz	27361	other files with extension
3180	.svg	32153	other files without extension

As indicated in Figure 6, the distribution of file sizes in this dataset is similar to DS2. The dataset comprises 284,523 files with a total size of approx. 10,050 MB. The average file size is 35.32 KB, the median is 1.29 KB, and the standard deviation is 1.08 MB.

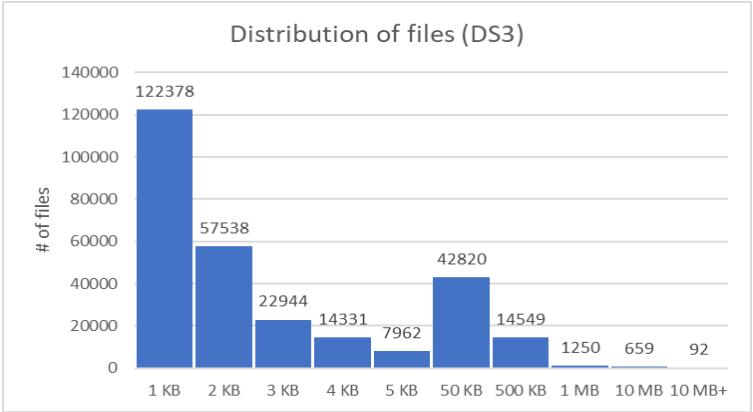

Figure 6. Distribution of file sizes in DS3

5. Results and evaluation

In this chapter, we present the results of our empirical analysis. Following the methodology outlined earlier, we measured each dataset – software pair.

The basis for our analysis includes the time required for creating the initial complete backup and the total size of the backup repository, as previously described.

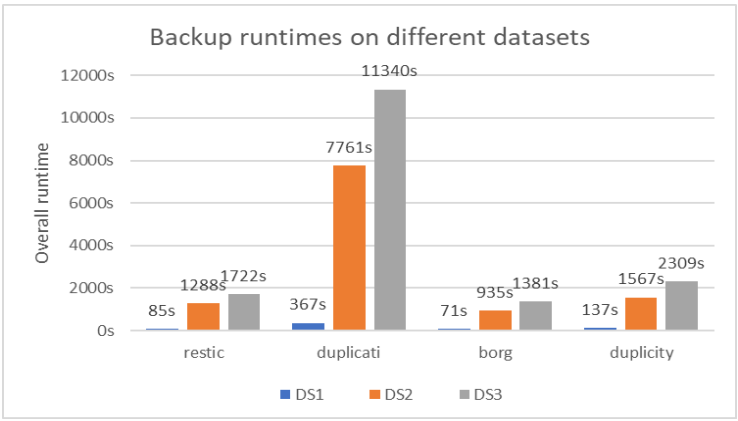

Figure 7. Overall runtimes of the backup tools on DS1-3 (the shorter, the better)

The average durations required for creating individual backups are illustrated in Figure 7.
In the case of DS1, there was an insignificant difference in the runtime of the various tools, which is likely attributed to the small size of the dataset and the partial compression applied to the content due to the PDF format.

In the case of DS2, the durations are also close to each other, with an extreme spike observed in the case of *duplicati*.

For the DS3 dataset, the same observations prove to be true, with the difference that *duplicati* exhibits an even larger deviation in the time required for backup execution. It took approximately 8.2 times longer to complete the task compared to the fastest tool, *borg*, while also approximately 4.9 times slower, than the second slowest *duplicity*.

In Figure 8, one can observe the optimization ratios achieved by the examined tools on datasets with varying levels of redundancy. A higher ratio indicates that the respective tool was able to compress the dataset to a smaller size in proportion to its original size. This is a measure of the efficiency of deduplication and compression processes.

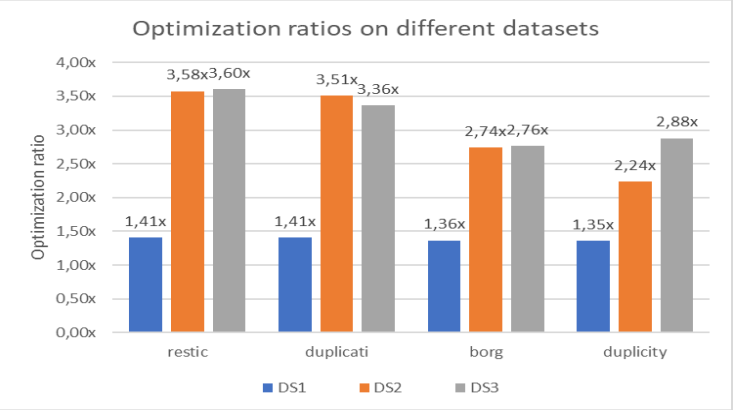

Figure 8. Optimization ratios reached by the backup tools on DS1-3 (the higher, the better)

In the case of DS1, the examined software performed nearly identically. This can be attributed to the balanced nature of the dataset and the compression inherent to the PDF format.

In DS2, *restic* and *duplicati* achieved nearly the same compression efficiency, while *borg* falls significantly behind them, and *duplicity* performs less efficiently. The top-ranked *restic* has nearly 60% advantage over the fourth-ranked *duplicity*.

In the case of DS3 the tools can be grouped into two categories: *restic* leading the way, followed closely by *duplicati*. *borg* and *duplicity* achieved similar results to each other but lagged behind the first-ranked tools.

An evaluation of the software can be fair when don't just the individual features are highlighted in isolation, but instead a more holistic approach be taken by

examining the measured results. In this regard, Figure 9 illustrates the trade-offs between storage requirements and execution speed in case of the examined tools.

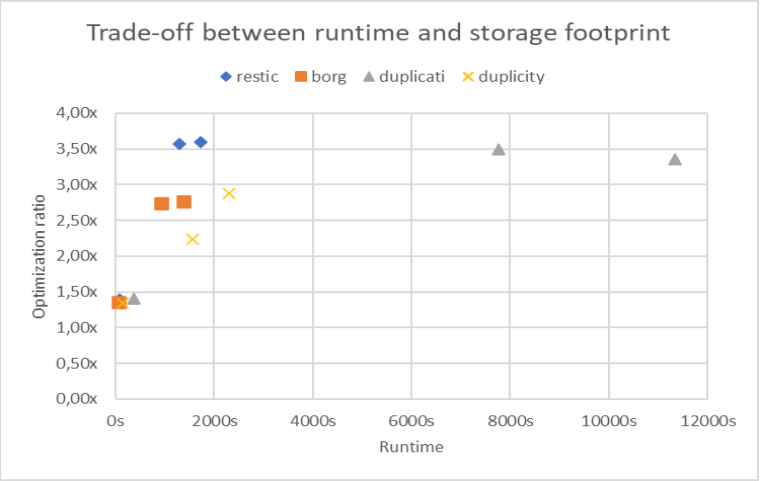

Figure 9. Trade-off between backup execution time and storage space of the final backup repository

This plot shows the relationship between runtime and storage footprint. Points on the left side of the diagram indicate shorter runtimes, while those on the right side indicate longer ones. Points in the upper part are associated with higher compression efficiency, while those in the lower part are associated with weaker redundancy reduction.

In the case of DS1, significant differences are not observed among the tested software. However, based on the measurements conducted on the DS2-3 datasets, the overall efficiency of individual tools can be distinguished.

In the case of *duplicati*, there is a significant increase in runtime, while the level of compression is nearly identical to the values measured with *restic*. However, user-friendliness can also play a significant role in the decision between these software options. While *restic* lacks an official GUI application and does not natively support backup scheduling, *duplicati* offers these features as a turnkey solution, potentially better meeting the needs of users with less technical knowledge, all while achieving one of the highest compression rates.

duplicity didn't achieve the best, but an acceptable compression rate, although it doesn't offer a time advantage and is less customizable compared to other solutions. *borg*'s optimization rate is also acceptable and, in terms of runtime it outperformed the other software in the study. In the case of *borg*, further experiments could be executed with fine-tuning of the backup process (e.g. using the ZStandard algorithm for compression). However, this study aims to measure the "as-is" performance, rather than detailed tuning.

restic's time requirement is slightly higher than *borg*'s, but its optimization ratio is significantly above the others.
Based on its characteristics, we recommend *restic* among the examined backup software for general use cases where system administrators want to avoid precise fine-tuning of the backup tools.

6. Conclusion

In this study, we evaluated the performance of four open-source backup tools, namely *borg*, *duplicati*, *duplicity*, and *restic*; across three diverse datasets. We focused on execution time and deduplication-compression efficiency, measured as the "optimization ratio".

Taking every measured value into consideration, *restic* emerged as a recommended choice for general use cases. *duplicati*, while achieving similar compression over a significantly longer period of time, offers user-friendliness with GUI support. *borg* showed potential for fine-tuning, and *duplicity*, while not the best in compression, provided an acceptable optimization ratio also.

In conclusion, *restic* is the most solid choice for users seeking an efficient open-source backup tool. However, the selection should consider user preferences, technical expertise, and specific use cases.

In the future, an interesting research direction could involve more in-depth quantitative software analysis, including incremental backups and file restoration. There are also prospects for further research in the optimization of backup tools, where different combinations of chunking algorithms, compression methods, and encryption techniques can be explored. Additionally, it is worth considering a comparison of the latest CDC algorithms as another potential direction for investigation.

References

[1] Wolff, J. (2023). Trends in Cybercrime During the COVID-19 Pandemic. In: *Beyond the Pandemic? Exploring the Impact of COVID-19 on Telecommunications and the Internet*. Emerald Publishing Limited, pp. 2015–227. https://doi.org/10.1108/978-1-80262-049-820231010

[2] Tekin, S. et al. (2023), Optimal data backup policies for information systems subject to sudden failure. *Journal of Quality in Maintenance Engineering*, Vol. 29, No. 2, pp. 338–355. https://doi.org/10.1108/JQME-01-2022-0009

[3] Qin, A., Xiao, M., Huang, B. & Zhang, X. (2022). Maze: A Cost-Efficient Video Deduplication System at Web-scale. In: *Proceedings of the 30th ACM International Conference on Multimedia*. https://doi.org/10.1145/3503161.3548145

[4] Prajapati, P. & Shah, P. (2022). A Review on Secure Data Deduplication: Cloud Storage Security Issue. *Journal of King Saud University – Computer and Information Sciences*, 34 (7), pp. 3996–4007. https://doi.org/10.1016/j.jksuci.2020.10.021

[5] Appaji Nag Yasa, G. & Nagesh, P. C. (2012). Space savings and design considerations in variable length deduplication. *ACM SIGOPS Operating Systems Review*, 46 (3), pp. 57–64. https://doi.org/10.1145/2421648.2421657

[6] Yoon, M. (2019). A constant-time chunking algorithm for packet-level deduplication. *ICT Express*, 5 (2), pp. 131–135. https://doi.org/10.1016/j.icte.2018.05.005

[7] Rajkumar, K. & Dhanakoti, V. (2022). Fuzzy-Dedup: A secure deduplication model using cosine-based Fuzzy interference system in cloud application. *Journal of Intelligent & Fuzzy Systems*, 43(3), 2819-2832. https://doi.org/10.3233/jifs-210511

[8] Jiang, T. et al. (2023). FuzzyDedup: Secure Fuzzy Deduplication for Cloud Storage. *IEEE Transactions on Dependable and Secure Computing*, 20 (3), pp. 2466–2483. https://doi.org/10.1109/tdsc.2022.3185313

[9] Rabin, M. O. (1981). *Fingerprinting by Random Polynomials*. Center of Research in Computer Technology, Technical Report.

[10] Xiaowei, M., Ketai, H., Xu, X. & Aijun, L. (2023). Research on an Incremental Backup Method Based on CAD Engineering Data File. *In 2023 IEEE 18th Conference on Industrial Electronics and Applications (ICIEA)*. https://doi.org/10.1109/iciea58696.2023.10241897

[11] Zhou, P., Wang, Z., Xia, W. & Zhang, H. (2022). UltraCDC: A Fast and Stable Content-Defined Chunking Algorithm for Deduplication-based Backup Storage Systems. *In 2022 IEEE International Performance, Computing, and Communications Conference (IPCCC)*. https://doi.org/10.1109/ipccc55026.2022.9894295

[12] Zhang, Y. et al. (2016). A Fast Asymmetric Extremum Content Defined Chunking Algorithm for Data Deduplication in Backup Storage Systems. *IEEE Transactions on Computers*, 1-1. https://doi.org/10.1109/tc.2016.2595565

[13] RFC 8878. Zstandard Compression and the 'application/zstd' Media Type. https://datatracker.ietf.org/doc/html/rfc8878

[14] Xia, W. et al. (2016). A Comprehensive Study of the Past, Present, and Future of Data Deduplication. *Proceedings of the IEEE*, 104 (9), pp. 1681–1710. https://doi.org/10.1109/jproc.2016.2571298

[15] Rclone. https://rclone.org/ (Accessed 09-10-2023).

[16] ISO 32000-2:2020 (PDF 2.0).

[17] *DOCX Transitional (Office Open XML), ISO 29500:2008-2016, ECMA-376, Editions 1-5*. https://www.loc.gov/preservation/digital/formats/fdd/fdd000397.shtml

[18] Sakamoto, Y. et al. (2015). Empirical study on effects of script minification and HTTP compression for traffic reduction. *In 2015 Third International Conference on Digital Information, Networking, and Wireless Communications (DINWC). IEEE*. https://doi.org/10.1109/dinwc.2015.7054230

[19] Król, K. (2020). Comparative Analysis of Selected Online Tools for JavaScript Code Minification: A Case Study of a Map Application. *Geomatics, Land Management and Landscape*, 2, pp. 119–129. https://doi.org/10.15576/gll/2020.2.119

[20] Valverde Cameselle, R., & Gonzalez Labrador, H. (2021). Addressing a billion-entries multi-petabyte distributed file system backup problem with cback: From files to objects. *EPJ Web of Conferences*, 251, 02071. https://doi.org/10.1051/epjconf/202125102071

[21] Restic Documentation. https://restic.readthedocs.io/en/latest/index.html (Accessed 09-10-2023).

[22] Borg Documentation. https://borgbackup.readthedocs.io/en/stable/ (Accessed 09-10-2023).

[23] GitHub. duplicate/duplicate. https://github.com/duplicati/duplicati (Accessed 09-10-2023).

[24] Duplicati. Choosing sizes in Duplicati. https://duplicati.readthedocs.io/en/latest/appendix-c-choosing-sizes-in-duplicati/ (Accessed 09-10-2023.

[25] Duplicity: Encrypted bandwidth-efficient backup. https://duplicity.us/ (Accessed 09-10-2023).