# Speed Kills: Revisiting Data Deduplication for Modern Storage Devices

Workshop paper

Rui Pedro Oliveira
*INESC TEC & U. Minho*
rui.v.oliveira@inesctec.pt

Tânia Esteves
*INESC TEC & U. Minho*
tania.esteves@inesctec.pt

João Paulo
*INESC TEC & U. Minho*
jtpaulo@inesctec.pt

***Abstract*—The rise of new, faster storage devices has introduced significant changes to the design and optimization of traditional storage systems. This paper examines how different storage hardware, ranging from HDDs to SSDs and Persistent Memory, influences the design and efficiency of modern deduplication systems. Through a literature consolidation and an empirical study on the performance implications of different fingerprinting methods and indexing configurations, we highlight important and non-trivial co-design choices that must be carefully considered when building next-generation deduplication systems.**

***Index Terms*—storage systems, disk devices, data deduplication**

## I. Introduction

New, faster storage devices have become mainstream in consumer and enterprise-grade applications in recent years. As the gap between CPU and I/O performance continues to diminish, one must carefully rethink and redesign traditional storage optimizations [1, 2]. One such optimization is data deduplication, which has been used from archival and backup systems to primary storage appliances towards reducing storage space requirements, increasing the lifespan of the devices, and even improving I/O performance [3].

Briefly, deduplication partitions stored content into chunks. If two chunks are the same, only a single copy is persisted, saving space. To achieve this, a fingerprint (i.e., a value that uniquely represents a chunk) must be computed and then kept in a data structure (an index) for efficient lookup. This process is computationally expensive, as, for example, fingerprinting is traditionally done using cryptographic hash functions [4]. As a result, deduplication designs must take into account not only I/O performance but also computational performance and, more crucially, the balance between them.

**Challenge and Related Work.** This delicate balance still requires careful study and systematization. Whilst previous work summarizes and classifies the different deduplication designs and techniques used in various use cases, it does not consider the new reality of fast storage devices and I/O frameworks [3, 4]. After these surveys were conducted, new deduplication systems using the Storage Performance Development Kit (SPDK) and Persistent Memory Development Kit (PMDK) frameworks and targeting NVMe SSD and Persistent Memory devices were proposed [5–9]. These new frameworks and storage mediums changed the established balance between I/O and computational performance, and, consequently, key fundamental design principles of deduplication solutions.

The main contribution of this paper is thus an empirical study and consolidation of the literature on the characteristics of new storage devices that significantly impact the design of deduplication. In more detail, our paper is the first to study:

*i*) distinct characteristics of HDD, SSD, and PMem devices (e.g., access time and patterns, read and write bandwidth asymmetry) and how these impact the design of core deduplication mechanisms such as fingerprint computation and duplicate detection;

*ii*) how compact index metadata structures can effectively handle the ever-increasing amount of stored data and the capacity of today's devices. Specifically, we analyze the behavior of partial indexes, an increasingly used in-memory metadata structure that keeps only a subset of fingerprints of stored chunks, under workloads with distinct duplication ratios and eviction policies.

In sum, we show that deduplication systems must be carefully co-designed with the intrinsic, and not always evident, characteristics of each device and I/O workloads, while pointing out open paths for future research work in the field.

## II. The Storage Landscape

In the past, the main storage medium was magnetic tape. Then, Hard Disk Drives (HDDs), with spinning magnetic disks and a moving read/write head for data storage and retrieval, became prevalent. Later, flash-based Solid-State Drives (SSDs) became the mainstream medium for performance-sensitive applications due to being faster, especially under random I/O workloads, and more energy efficient than HDDs [10].

As flash technology improved, the existing SATA protocol used for communication between the device and the CPU became a bottleneck. In response, the Non-Volatile Memory Express (NVMe) protocol was developed, which uses a faster transport protocol underneath (PCIe instead of AHCI), allowing for higher parallel data processing and lower latency [11].

Recently, a new storage medium referred to as Persistent Memory (PMem) was developed. It is a non-volatile, low-latency, byte-addressable memory with densities greater than DRAM, and can be used as a regular storage device or as

main memory [12]. Despite being discontinued, it remains relevant research-wise, as mediums with similar properties are currently being developed [13, 14].

*Discussion.* Not only did devices become faster, but their performance characteristics have also changed over time, affecting how applications should be implemented and tuned for different storage mediums.

HDDs are known to exhibit poor random I/O performance due to their mechanical design (i.e., arm and head movement when seeking across random disk sectors). In contrast, sequential performance is significantly better, and the bandwidth asymmetry between reads and writes is negligible [15].

On SSD devices, random writes tend to be slower than sequential ones, as they often trigger erasure operations and can lead to write amplification [16]. Meanwhile, read performance remains consistent between random and sequential access, since SSDs have no seek time (as in HDDs) [17]. Notably, read bandwidth is typically higher than the write counterpart ($\approx$1.06$\times$ for SATA SSDs, $\approx$2.1$\times$ for NVMe SSDs).[1]

On PMem devices, sequential reads are faster than random ones, mainly due to their built-in read-ahead mechanisms. As writes do not benefit from these mechanisms, sequential and random write workloads perform similarly. Unlike in the other two storage mediums, PMem's read bandwidth is considerably larger (up to 3$\times$) than the write one [6, 17].

## III. Data Deduplication

Data deduplication works by identifying and removing duplicate data present in storage systems, therefore enabling space savings. An example flow for data deduplication is shown in Figure 1. To find duplicates, this technique must first partition stored data into smaller pieces, called chunks, that act as the deduplication unit (Figure 1-❶). There are multiple approaches to partitioning (*i.e.*, chunking) data, from whole-file chunks to fixed and variable-sized ones. After defining the unit of deduplication, one must be able to identify duplicate chunks. As comparing byte-wise every two chunks (*i.e.*, byte-to-byte comparison) is computationally expensive, the content of each chunk is commonly identified by a fingerprint (Figure 1-❷), a unique digest which is usually the result of hashing the chunk's content with a hash function.

To further speed the identification of duplicate chunks, fingerprints are indexed in a metadata structure (Figure 1-❸), commonly referred to as the deduplication index. The index tracks all the fingerprints of stored chunks, allowing it to quickly identify a potential duplicate candidate for a newly written chunk. When a duplicate chunk is found, it means that the exact same content was previously persisted, and the new chunk does not need to be actually stored (Figure 1-❹). Instead, the logical address of this new chunk will point to the physical location on disk where the duplicate content resides. Having multiple logical chunk addresses pointing to the same

[1]The devices chosen for this comparison are the Hynix HFS480G3H2X069N (Se5110 Series) SATA SSD [18] and the Dell Express Flash P5500/P5600 NVME SSD [19].

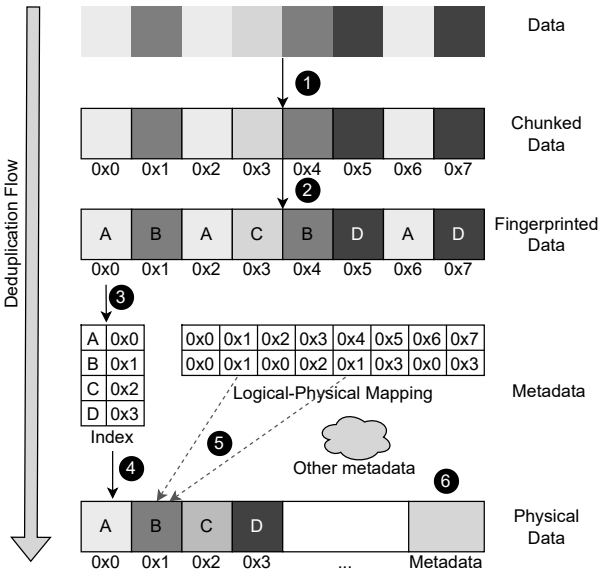


Fig. 1: High-level overview of data deduplication

physical address is the basic mechanism for content sharing (Figure 1-❺). The mapping between logical and physical views of the system is kept alongside the index as metadata to enable transparent access from applications. Furthermore, other structures can be kept for optimization purposes (e.g., how many logical references each physical chunk address has, free space management). To ensure crash consistency, this metadata must also be persistently stored (Figure 1-❻).

These steps can be done at different stages of I/O requests. Inline deduplication refers to systems that perform all the previous steps in the critical I/O path of requests.[2] Alternatively, offline deduplication defers steps ❷ to ❻ to asynchronous background tasks. This decision avoids an extra latency penalty on foreground I/O requests at the cost of temporarily storing duplicate content at the storage system.

## IV. Study on Deduplication and Devices Co-design

As storage devices have different performance characteristics, deduplication designs have adapted accordingly.

**Deduplication over HDDs.** A considerable body of work on deduplication emerged while the mainstream storage medium was spinning hard drives [4]. As computation was significantly faster than I/O, cryptographic hashes were the norm for chunk fingerprinting, even when using inline deduplication. Additionally, as reading metadata and the index from disk carried a significant performance penalty, many inline systems opted to cache a subset of fingerprints in RAM or use Bloom filters to speed up the index lookup [20].

As hard drives were sensitive to fragmentation, worsened by deduplication due to changing the chunks' physical location,

[2]Metadata persistence can be deferred to a later time by grouping together multiple I/Os in a single operation.

there was a concern to preserve the on-disk layout of data by deduplicating and storing contiguous chunks together [4, 21].

**Deduplication over SSDs.** The first works on deduplication for SSDs were designed and implemented as part of the device controller, exploiting its hardware and firmware. Many of these solutions used the Flash Translation Layer (FTL) component to maintain the logical-physical metadata mapping with minimal performance overhead. Similarly, other SSD components for wear leveling and garbage collection were adapted to manage references for duplicate pages [22–24].

As the DRAM included on SSDs is typically small, these devices do not have sufficient space for holding the full deduplication index. To mitigate this issue, some systems have adopted a best-effort deduplication approach. This involves either storing a partial index in DRAM, keeping the entire index in the DRAM of the host OS, or using a smaller section of NVRAM embedded inside the SSD [22–24].

While deduplication for NVMe SSDs was also explored at the OS-level, its design highly resembles the ones followed by more traditional solutions over HDDs [5].

**Deduplication over PMem.** In traditional deduplication systems, cryptographic hash functions were chosen as the main chunk fingerprinting approach due to their collision resistance (i.e., very low probability of two chunks with distinct content having the same hash digest) [4]. However, their computation is expensive, especially when done in the critical path of I/O requests to PMem devices, which exhibit significantly lower latency than HDD and SSD mediums. As a result, deduplication systems for PMem started adopting non-cryptographic hash functions (e.g., xxHash, CRC-32) [6, 8, 9].

These hash functions, however, do not provide strong resistance against collisions, unlike the cryptographic ones. This means that, whenever two chunks are identified as duplicates (i.e., their hashes match), their actual content must be compared, typically with a byte-to-byte comparison, to ensure that they indeed have the exact same content [6, 8, 9].

*Discussion.* The existing literature, and its brief overview provided so far in this paper, highlight interesting design decisions based on the type of device being targeted. However, this information is typically scattered across distinct works, and there is no unified view of how the design of key deduplication mechanisms is impacted by using different disks ranging from HDDs to SSDs and PMem. Next, we showcase the importance of such study by focusing on two important features of deduplication: chunk fingerprinting and indexing.

### A. Chunk Fingerprinting

The choice of hash function for computing chunk fingerprints presents a trade-off between I/O performance, metadata size (i.e., larger fingerprints will lead to bigger deduplication indexes), and collision resistance. Table I shows a comparison between the following four hash functions:

- **SHA-512:** a cryptographic hash function designed by the NSA, used in a variety of contexts, such as digital signatures and secure communication protocols. We chose the 512-bit variant as it is more secure and exhibits similar performance to the 256-bit variant [25];
- **Blake3:** a recent, parallelizable cryptographic hash function faster than traditional ones. We opt for its default 256-bit digest size, as the larger digests do not lead to better performance or collision resistance [26];
- **CRC-32:** a non-cryptographic hash function commonly used for error detection. We use the 32-bit variant as it is used in other deduplication systems [8];
- **xxHash:** a fast non-cryptographic hash algorithm, working at RAM speed limit [27]. We use the 64-bit version as it is the one used in previous deduplication work [6].

The latency and throughput of these functions were evaluated on a system with two Intel® Xeon® Gold 6342 CPUs and 192GiB of DDR4-3200 RAM by computing 100M hashes of blocks of 4KiB, a commonly used size for deduplication systems. Collision probability results were computed assuming a perfect hash distribution, based on the Birthday Paradox [28].

Results show Blake3 achieves 5.97× more throughput than SHA-512, with half the digest size, while keeping a negligible collision probability. For non-cryptographic hashes, xxHash shows 1.85× more throughput than CRC-32 and a lower collision probability, though it doubles the digest size. Compared to SHA-512, xxHash improves throughput by 20× and produces smaller digests, albeit with an increased risk of collisions.

Based on the data from Table I, we study the performance impact of these hash functions on deduplication. Figure 2 plots the overhead of computing the different functions in the critical I/O path, measured as the percentage of the total I/O latency spent on fingerprinting, assuming storage devices operating at nominal capacity with varying bandwidth values. For reference, we label bandwidth values for existing storage devices (vertical dashed lines in Figure 2): 153MiB/s for a HDD [29], 505MiB/s for a SATA SSD [18], 1716MiB/s for an NVMe SSD [19], and 2861MiB/s for a PMem device [30]. As multiple hash computations can run in parallel, we also plot the overhead relative to how many threads are performing I/O, assuming disk bandwidth is equally distributed amongst them.

With 8 I/O threads, the overhead of using xxHash instead of SHA-512 decreases significantly, especially for faster mediums: from 3% to 0.16% on the HDD, from 10% to 0.52% on the SATA SSD, from 26% to 1.75% on the NVMe SSD, and from 37.4% to 2.9% on PMem. When considering today's storage appliances with multiple devices or anticipating future devices reaching, for instance, 5 GiB/s of bandwidth, the overhead of SHA-512 becomes even more noticeable. With 8 threads, it would rise to 51.7%, compared to 15.2% for Blake3, 9.0% for CRC-32, and 5.1% for xxHash.

As the number of threads increases, the available disk bandwidth per thread becomes smaller, and consequently, the overhead of computing a hash decreases. This is more noticeable for faster devices and slower hashes. For example, the overhead of SHA-512 on the PMem device drops by 45 *pp* when increasing threads from 1 to 8, whereas xxHash's overhead in the same scenario only decreases by 16 *pp*. Notably, increasing the number of threads can lead to higher contention

TABLE I: Comparison of SHA-512, Blake3, CRC-32 and xxHash hash functions

| | SHA-512 | Blake3 | CRC-32 | xxHash |
|---|---|---|---|---|
| Cryptographic | Yes | Yes | No | No |
| Latency (ns) | $6535 \pm 1579$ | $1095 \pm 700$ | $603 \pm 510$ | $325 \pm 429$ |
| Throughput (MiB/s) | $598 \pm 7$ | $3569 \pm 58$ | $6473 \pm 60$ | $12001 \pm 145$ |
| Digest Size (B) | 64 | 32 | 4 | 8 |
| Hashes needed for 50% chance of collisions | $1.4 \times 10^{77}$ | $4.0 \times 10^{38}$ | $7.7 \times 10^{4}$ | $5.1 \times 10^{9}$ |

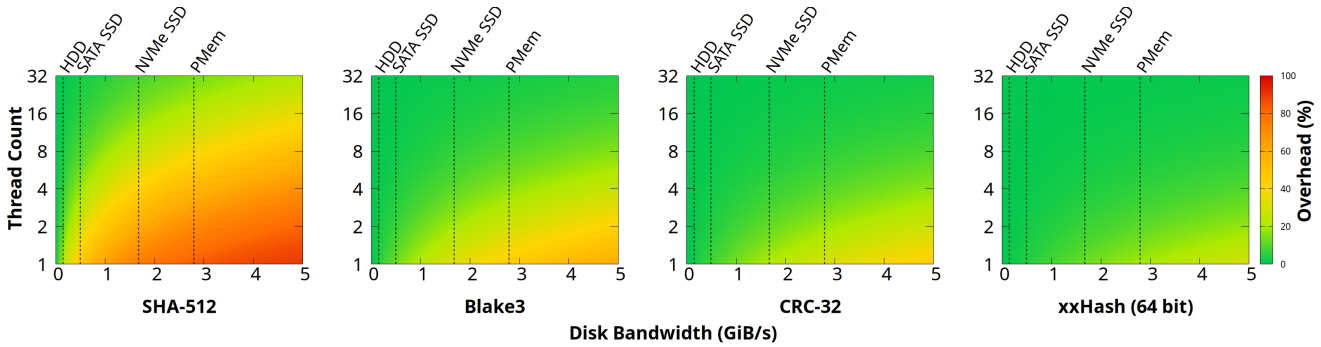


Fig. 2: Scalability of SHA-512, Blake3, CRC-32 and xxHash hash functions

during deduplication metadata management operations. This is because metadata synchronization has to occur across more threads, resulting in more contention in critical regions, the effects of which are not accounted for in these plots.

All in all, cryptographic hashes provide better collision resistance than non-cryptographic hashes but also have a considerable performance overhead. This overhead is noticeable on NVMe SSDs and becomes even more pronounced on PMem devices, where functions like SHA-512 take too long to compute within the critical I/O path.

> **Takeaway 1.** *Traditional cryptographic hashes like SHA-512 incur negligible overhead on slow (HDD) devices, but prohibitive overhead on fast (NVMe SSDs, PMem) mediums. While faster cryptographic functions like Blake3 are generally efficient on SSDs, PMem demands an even faster, non-cryptographic hash function to maintain performance in latency-sensitive I/O operations.*

However, the previous analysis is insufficient for choosing the appropriate fingerprinting method, as the cost of performing byte-to-byte comparison in non-cryptographic hash functions must also be taken into account. This cost is not trivially calculated, as it depends on multiple factors:

- ***I/O workload*** – Workloads with a higher volume of duplicates will require more byte comparisons, as the likelihood of fingerprint matches increases (e.g., duplicate chunks sharing the same fingerprint);
- ***Collision resistance of hash function*** – The less susceptible a hash function is to collisions, the less byte comparisons are required to distinguish different chunks that may end up with the same hash signature;
- ***Read/write asymmetry of device*** – Devices with high read/write asymmetry can fetch content quicker than they can write it, alleviating the overhead of doing an additional read from disk under a write operation. It is the reason PMem systems can successfully leverage non-cryptographic hash functions. However, mechanisms used to reduce the impact of reading from disk, like prefetching and speculative reads, must be taken into account when evaluating the cost of byte comparison [6, 9].

In sum, non-cryptographic hash functions are worthwhile if these factors mean the amortized cost of computing the non-cryptographic hash function, fetching content when hash collisions are detected, and performing in-memory byte-to-byte comparison is less than the cost of computing a cryptographic hash function for every intercepted write operation.

> **Takeaway 2.** *The trade-off of using a non-cryptographic hash function and byte-to-byte comparison versus using a cryptographic hash function is not trivial to evaluate, as it depends on the I/O workload the system is subject to, the collision resistance of the chosen hash function, and the read/write asymmetry of the target device.*

> **Takeaway 3.** *To opt for a non-cryptographic hash function, deduplication systems must evaluate its cost based on the workload and device characteristics, to see if it is less than that of computing a cryptographic hash function.*

### *B. Indexing*

As the scale of stored data increases, deduplication indexes become too large to fully fit in RAM. As an example, let us assume a system storing 10TiB of data and performing deduplication in fixed-sized chunks of 4KiB. If the index has a size of 20B per entry (8B for xxHash, 8B for location on disk, 4B for reference count), the total size of the index will

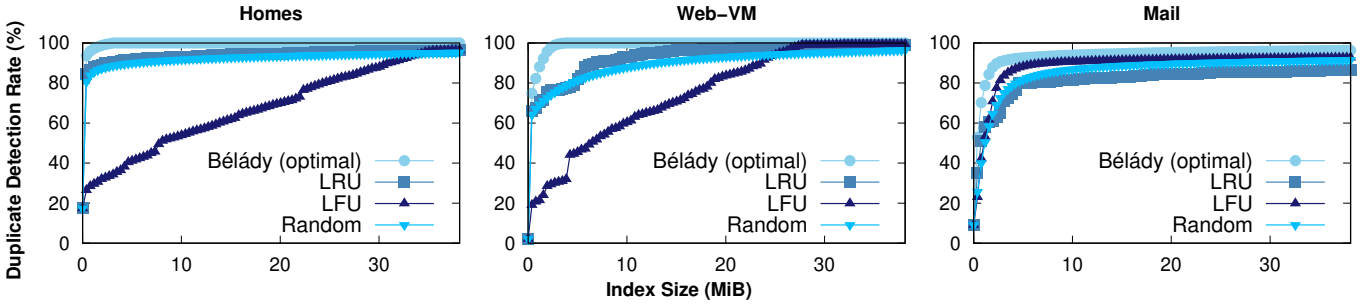


Fig. 3: Impact of index size and cache eviction policy on the percentage of correctly identified duplicates

be 50GiB. As such, to perform global deduplication across all stored chunks, systems must persist the index on the device while caching a subset of on-disk entries in RAM to minimize the performance penalty of fully reading index entries from disk [4]. Another approach, used by the literature, is to opt for partial deduplication, *i.e.*, keeping a subset of index entries solely in RAM and using only these to find duplicates [9, 21].

When following any of these two approaches, determining which entries to keep in RAM and which to evict requires careful consideration, as there are many possible eviction policies, like Least-Recently-Used (LRU), Least-Frequently-Used (LFU), and Random Replacement.

We study how these policies affect the space savings of a partial deduplication solution under distinct index sizes (i.e., number of entries kept at the partial in-memory deduplication index). Further, the above policies are compared with a theoretically optimal eviction strategy under three I/O workloads [31].[3] The workloads are three publicly available, real-world block traces collected at the Florida International University (FIU) Computer Science Department, which have been highly used in academia to evaluate the real-world performance of deduplication systems [5, 32]. These replicate the behaviour of an email server (*mail*), a file server (*homes*), and a virtual machine running two web servers (*web-vm*). To compute the optimal sequence of evictions, we use Bélády's cache replacement algorithm, which is optimum given information on the workload behaviour [33].[4]

As Figure 3 shows, small index sizes (0.03MiB, 0.65MiB, and 1.23MiB for the *homes*, *web-vm*, and *mail* workloads, respectively) are enough in theory to identify over 80% of the duplicates in all workloads (assuming 20B/entry).[5] In practice, indexes need to be larger, as LRU requires 0.20, 4.80, and 2.54MiB to achieve the same gains. Surprisingly, the random eviction policy performs only slightly worse than LRU, requiring 0.35, 4.38, and 4.29MiB to achieve the same 80% target. LFU is the worst-performing policy (24.35, 18.50, 4.57MiB, respectively). One explanation for its poor performance is its susceptibility to cache solidification, a phenomenon where some cache entries are never evicted, despite not being used for a long time, due to having more references than newer, hotter entries. This leads to missed deduplication opportunities [9].

> **Takeaway 4.** *Partial indexes may be enough to achieve good deduplication performance whilst maintaining a low memory footprint. However, their efficiency is affected by I/O workload's properties and the chosen eviction policy.*

## V. Future Direction

Besides solidifying existing knowledge on the co-design of deduplication systems and storage devices, our study and takeaways also help envision future trends that may shape the field. We next enumerate some of them.

**New Devices and Protocols.** Even though Intel discontinued Optane PMem devices [13], similar devices will appear, and technologies like NVMe over Fabrics and CXL promise remote access to storage devices faster than ever. This will probably lead to the redesign of distributed deduplication solutions, similarly to what we have seen for local designs.

**Computational Storage.** Computational storage refers to the offloading of computational load to storage devices equipped with a processing unit. Storage devices are already used inside Infrastructure Processing Units (IPUs), and can even communicate in the network to achieve distributed storage [34]. Furthermore, the NVMe specification recently defined this concept, creating a standard mechanism to execute programs inside NVMe devices [35]. Such will allow redefining what deduplication mechanisms can run closer to the devices towards improving overall performance.

In summary, these trends mean the gap between CPU and I/O performance will keep reducing, making high-performance deduplication more challenging.

***Computation on the critical I/O path.*** As computation in the critical I/O path becomes more expensive, the use of cryptographic hash functions will be prohibited when dealing with latency-sensitive workloads. This will make non-cryptographic hashes the default fingerprinting scheme for inline deduplication. In scenarios where byte comparison is too costly, offline deduplication with cryptographic hash functions will probably see broader adoption.

[3]The random policy results are averages of 10 runs with different seeds, and exhibit a maximum standard deviation of less than 0.01 percentage points.

[4]This representation is a slight simplification of the real problem, as we are ignoring the impact of block rewrites in deduplication performance.

[5]The plot shows the percentage of duplicate content detected relative to the total volume of duplicate content in the trace. A value of 100% indicates perfect identification of all duplicates, not that all content was duplicated.

***Indexing and Metadata Management.*** As disks keep growing in size and performance, the indexes will still not fit in RAM. However, the performance penalty of looking up fingerprints in a persisted index (and metadata) will decrease, which may lead to more solutions where these are persisted entirely in the target medium, reducing the RAM footprint associated with deduplication, and making it viable in resource-limited environments. However, caching index entries in RAM might still be beneficial to speed up fingerprint lookups.

## ACKNOWLEDGEMENTS

We thank the anonymous reviewers for their constructive feedback, which helped to improve the paper. This work is co-funded by the European Regional Development Fund (ERDF) through the NORTE 2030 Regional Programme under Portugal 2030, within the scope of the project BCD.S+M, reference 14436 (NORTE2030-FEDER-00584600). This work is co-financed by Component 5 - Capitalization and Business Innovation, integrated in the Resilience Dimension of the Recovery and Resilience Plan within the scope of the Recovery and Resilience Mechanism (MRR) of the European Union (EU), framed in the Next Generation EU, for the period 2021 - 2026, within project ATE, with reference 56.

## REFERENCES

[1] L. Vogel, A. Van Renen, S. Imamura, J. Giceva, T. Neumann, and A. Kemper, "Plush: A write-optimized persistent log-structured hash-table," *VLDB Endowment*, vol. 15, no. 11, pp. 2895–2907, 2022.

[2] B. Lepers and W. Zwaenepoel, "Johnny cache: the end of {DRAM} cache conflicts (in tiered main memory systems)," in *USENIX Symposium on Operating Systems Design and Implementation*, 2023, pp. 519–534.

[3] W. Xia, H. Jiang, D. Feng, F. Douglis, P. Shilane, Y. Hua, M. Fu, Y. Zhang, and Y. Zhou, "A comprehensive study of the past, present, and future of data deduplication," *Proceedings of the IEEE*, vol. 104, no. 9, pp. 1681–1710, 2016.

[4] J. Paulo and J. Pereira, "A survey and classification of storage deduplication systems," *ACM Computing Surveys*, vol. 47, no. 1, pp. 1–30, 2014.

[5] M. Miranda, T. Esteves, B. Portela, and J. Paulo, "S2dedup: Sgx-enabled secure deduplication," in *ACM international conference on systems and storage*, 2021, pp. 1–12.

[6] J. Qiu, Y. Pan, W. Xia, X. Huang, W. Wu, X. Zou, S. Li, and Y. Hua, "{Light-Dedup}: A light-weight inline deduplication framework for {Non-Volatile} memory file systems," in *USENIX Annual Technical Conference*, 2023, pp. 101–116.

[7] H. Kwon, Y. Cho, A. Khan, Y. Park, and Y. Kim, "Denova: Deduplication extended nova file system," in *IEEE International Parallel and Distributed Processing Symposium*. IEEE, 2022, pp. 1360–1371.

[8] P. Zuo, Y. Hua, M. Zhao, W. Zhou, and Y. Guo, "Improving the performance and endurance of encrypted non-volatile main memory through deduplicating writes," in *Annual IEEE/ACM International Symposium on Microarchitecture*. IEEE, 2018, pp. 442–454.

[9] C. Du, Z. Lin, S. Wu, Y. Chen, J. Wu, S. Wang, W. Wang, Q. Wu, and B. Mao, "FSDedup: Feature-Aware and Selective Deduplication for Improving Performance of Encrypted Non-Volatile Main Memory," *ACM Transactions on Storage*, vol. 20, no. 4, pp. 1–33, 2024.

[10] S. Swanson and A. M. Caulfield, "Refactor, reduce, recycle: Restructuring the i/o stack for the future of storage," *Computer*, vol. 46, no. 8, pp. 52–59, 2013.

[11] Q. Xu, H. Siyamwala, M. Ghosh, T. Suri, M. Awasthi, Z. Guz, A. Shayesteh, and V. Balakrishnan, "Performance analysis of NVMe SSDs and their implication on real world databases," in *ACM International Systems and Storage Conference*, 2015, pp. 1–11.

[12] "What is Persistent Memory?" https://www.snia.org/education/what-is-persistent-memory, accessed: 2025-06-12.

[13] A. Shilov, "Intel schedules the end of its 200-series Optane memory DIMMs — shipments to draw to an end in late 2025," https://www.tomshardware.com/pc-components/ssds/intel-schedules-the-end-of-its-200-series-optane-memory-dimms-shipments-to-draw-to-an-end-in-late-2025, accessed: 2025-06-12.

[14] X. K. Song, S. Liu, and G. P. on Sep 20, "Persistent memory – A new hope," https://www.sigarch.org/persistent-memory-a-new-hope/, Sep 2022, accessed: 2025-06-12.

[15] T. I. Papon, "Enhancing data systems performance by exploiting ssd concurrency & asymmetry," in *IEEE International Conference on Data Engineering (ICDE)*, 2024, pp. 5644–5648.

[16] S. Lee, S. Moon, and C. Kim, "Rflru: A replacement policy for flash memory storage systems," *Engineering Letters*, vol. 22, no. 4, pp. 290–295, 2014. [Online]. Available: https://www.engineeringletters.com/issues_v22/issue_4/EL_22_4_02.pdf

[17] S. Gugnani, A. Kashyap, and X. Lu, "Understanding the idiosyncrasies of real persistent memory," *VLDB Endowment*, vol. 14, no. 4, p. 626–639, Dec. 2020. [Online]. Available: https://doi.org/10.14778/3436905.3436921

[18] "Hynix HFS480G3H2X069N SE5110 Series Dell Oem 480GB SATA 6Gbps TLC 2.5 SSD," https://www.serversupply.com/SSD/SATA-6GBPS/480GB/HYNIX/HFS480G3H2X069N_344834.htm, accessed: 2025-07-02.

[19] "Dell Express Flash P5500 and P5600," https://dl.dell.com/manuals/all-products/esuprt_data_center_infra_int/esuprt_data_center_infra_storage_adapters/dell-poweredge-exp-fsh-nvme-pcie-ssd_deployment-guide4_en-us.pdf, Tech. Rep., accessed: 2025-07-02.

[20] F. Guo and P. Efstathopoulos, "Building a high-performance deduplication system," in *USENIX Annual Technical Conference*, 2011.

[21] K. Srinivasan, T. Bisson, G. R. Goodson, and K. Voruganti, "idedup: latency-aware, inline data deduplication for primary storage." in *USENIX Conference on File and Storage Technologies*, vol. 12, 2012, pp. 1–14.

[22] F. Chen, T. Luo, and X. Zhang, "{CAFTL}: A {Content-Aware} flash translation layer enhancing the lifespan of flash memory based solid state drives," in *USENIX Conference on File and Storage Technologies*, 2011.

[23] Y. Zhou, Q. Wu, F. Wu, H. Jiang, J. Zhou, and C. Xie, "{Remap-SSD}: Safely and efficiently exploiting {SSD} address remapping to eliminate duplicate writes," in *USENIX Conference on File and Storage Technologies*, 2021, pp. 187–202.

[24] Y. Wen, X. Zhao, Y. Zhou, T. Zhang, S. Yang, C. Xie, and F. Wu, "Eliminating storage management overhead of deduplication over ssd arrays through a hardware/software co-design," in *Proceedings of the 29th ACM International Conference on Architectural Support for Programming Languages and Operating Systems*, 2024, pp. 320–335.

[25] Q. H. Dang and S. Turner, "Sha-512/224 and sha-512/256," https://datatracker.ietf.org/doc/html/draft-dang-turner-sha-512-224-256-00, May 2013, accessed: 2025-07-09.

[26] "The Official Rust and C Implementations of the BLAKE3 Cryptographic Hash Function," https://github.com/BLAKE3-team/BLAKE3, accessed: 2025-06-17.

[27] "xxHash - Extremely Fast Non-cryptographic Hash Algorithm," https://xxhash.com/, accessed: 2025-06-17.

[28] S. Goldwasser and M. Bellare, "Lecture notes on cryptography," *Summer course "Cryptography and computer security" at MIT*, pp. 249–250, 2005.

[29] "Seagate barracuda pro st500lm034 500gb sata hard drive," https://www.disctech.com/Seagate-Barracuda-Pro-ST500LM034-500GB-SATA-Hard-Drive, accessed: 2025-07-02.

[30] P. Alcorn, "Optane DIMM pricing." https://www.tomshardware.com/news/intel-optane-dimm-pricing-performance,39007.html, accessed: 2025-07-02.

[31] "FIU IODedup," https://iotta.snia.org/traces/block-io/391, accessed: 2025-06-17.

[32] A. Godavari, C. Sudhakar, and T. Ramesh, "Hybrid deduplication system—a block-level similarity-based approach," *IEEE Systems Journal*, vol. 15, no. 3, pp. 3860–3870, 2020.

[33] L. A. Belady, "A study of replacement algorithms for a virtual-storage computer," *IBM Systems journal*, vol. 5, no. 2, pp. 78–101, 1966.

[34] B. Walker, "Spdk as ipu firmware - part 1," https://spdk.io/dev/2023/04/20/memory-domains-1/, 2023, accessed: 2025-06-17.

[35] *Computational Programs Command Set Specification*, NVM Express®, 3 2025, rev. 1.1.