An Enterprise-Grade Open-Source Data Reduction Architecture for All-Flash Storage Systems

MOHAMMADAMIN AJDARI, Sharif University of Technology, Iran
PATRICK RAAF, Johannes Gutenberg University Mainz, Germany
MOSTAFA KISHANI, Sharif University of Technology, Iran
REZA SALKHORDEH, Johannes Gutenberg University Mainz, Germany
HOSSEIN ASADI*, Sharif University of Technology, Iran
ANDRÉ BRINKMANN, Johannes Gutenberg University Mainz, Germany

All-flash storage (AFS) systems have become an essential infrastructure component to support enterprise applications, where sub-millisecond latency and very high throughput are required. Nevertheless, the price per capacity of *solid-state drives* (SSDs) is relatively high, which has encouraged system architects to adopt *data reduction* techniques, mainly *deduplication* and *compression*, in enterprise storage solutions. To provide higher reliability and performance, SSDs are typically grouped using *redundant array of independent disk* (RAID) configurations. Data reduction on top of RAID arrays, however, adds I/O overheads and also complicates the I/O patterns redirected to the underlying backend SSDs, which invalidates the best-practice configurations used in AFS. Unfortunately, existing works on the performance of data reduction do not consider its interaction and I/O overheads with other enterprise storage components including SSD arrays and RAID controllers.

In this paper, using a real setup with enterprise-grade components and based on the open-source data reduction module RedHat VDO, we reveal novel observations on the performance gap between the state-of-the-art and the optimal all-flash storage stack with integrated data reduction. We therefore explore the I/O patterns at the storage entry point and compare them with those at the disk subsystem. Our analysis shows a significant amount of I/O overheads for guaranteeing consistency and avoiding data loss through data journaling, frequent small-sized metadata updates, and duplicate content verification. We accompany these observations with cross-layer optimizations to enhance the performance of AFS, which range from deriving new optimal hardware RAID configurations up to introducing changes to the enterprise storage stack. By analyzing the characteristics of I/O types and their overheads, we propose three techniques: **(a)** application-aware lazy persistence, **(b)** a fast, read-only I/O cache for duplicate verification, and **(c)** disaggregation of block maps and data by offloading block maps to a very fast persistent memory device. By consolidating all proposed optimizations and implementing them in an enterprise AFS, we show 1.3× to 12.5× speedup over the baseline AFS with 90% data reduction, and from 7.8× up to 57× performance/cost improvement over an optimized AFS (with no data reduction) running applications ranging from 100% read-only to 100% write-only accesses.

CCS Concepts: • **Information systems → Storage architectures**; **RAID**.

*Corresponding Author, work has been performed as a visiting professor at Johannes Gutenberg University Mainz, Germany.

Authors' addresses: Mohammadamin Ajdari, Sharif University of Technology, Tehran, 11155-11365, Iran, m.ajdari@sharif.edu; Patrick Raaf, Johannes Gutenberg University Mainz, Mainz, Germany, raaf@uni-mainz.de; Mostafa Kishani, Sharif University of Technology, Azadi, Tehran, 11155-11365, Iran, mostafa.kishani@sharif.edu; Reza Salkhordeh, Johannes Gutenberg University Mainz, Mainz, Germany, rsalkhor@uni-mainz.de; Hossein Asadi, Sharif University of Technology, Azadi, Tehran, 11155-11365, Iran, asadi@sharif.edu; André Brinkmann, Johannes Gutenberg University Mainz, Mainz, Germany, brinkman@uni-mainz.de.

2476-1249/2022/6-ART30 $15.00
https://doi.org/10.1145/3530896

Additional Key Words and Phrases: All-Flash Storage Systems, Deduplication, Performance Evaluation, Compression, RAID, Solid-State Drives

ACM Reference Format:
Mohammadamin Ajdari, Patrick Raaf, Mostafa Kishani, Reza Salkhordeh, Hossein Asadi, and André Brinkmann. 2022. An Enterprise-Grade Open-Source Data Reduction Architecture for All-Flash Storage Systems. *Proc. ACM Meas. Anal. Comput. Syst.* 6, 2, Article 30 (June 2022), 27 pages. https://doi.org/10.1145/3530896

1 INTRODUCTION

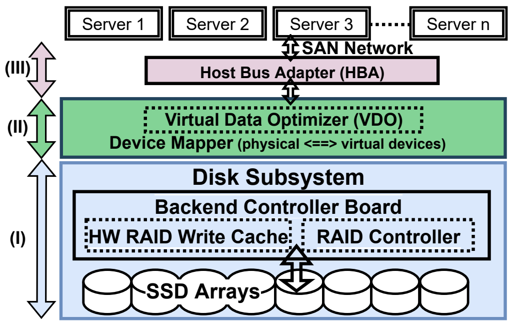
All-flash storage (AFS) systems are increasingly being used as primary storage infrastructure in enterprise applications, where very low latency (less than a millisecond) and very high throughput (hundreds of thousands of I/Os per second) are strictly required [27, 36]. To meet the increasing demand for higher performance, reliability, and capacity, solid-state drives (SSDs) in all-flash storage systems are typically grouped in arrays and configured as redundant arrays of independent disks (RAID), controlled by either a software or hardware RAID controller, as depicted in Fig. 1a (I). The significantly higher cost of SSDs compared to *hard disk drives* (HDDs), however, is a major barrier to use all-flash storage systems for a wide range of applications, where many *petabytes* (PBs) of user data are continuously stored and accessed in large-scale systems.

To take advantage of all-flash storage systems in applications with capacity requirements of many PBs, *data reduction* (DR) techniques have been offered in recent years to minimize storage costs [4, 28, 40, 73, 75]. Data reduction mainly consists of *deduplication* and *compression*. Deduplication identifies blocks of the same content and replaces them with references to the already persisted copy, instead of storing them again. As a complement to deduplication, remaining data blocks go through compression, where similar content is eliminated at byte granularity. Employing deduplication and compression together can reduce the data footprint by up to 90% in primary storage [13, 16].

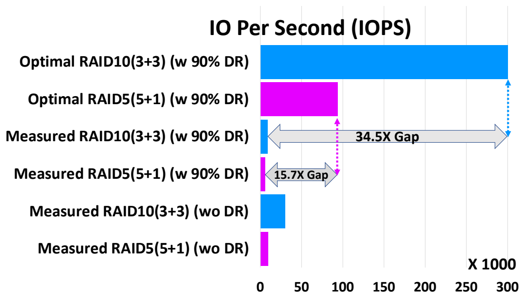
Several open-source data reduction projects have been introduced in the past few years; however, these techniques *either* are tuned for backup storage (e.g., OpenDedup [57]) instead of primary storage; *or* are implemented at the file system level; *or* suffer from a lack of industry support (e.g., dm-dedup [60]). These shortcomings have prevented the implementation of such open-source projects in enterprise primary storage systems. The recent open-source data reduction project *virtual data optimizer* (VDO) [22] overcomes these shortcomings by a) including a block device implementation, b) targeting all-flash primary storage as a major application domain, and c) providing industry support from RedHat. VDO can furthermore be easily stacked on top of the Linux *device mapper* layer (Fig. 1a (II)) and therefore has great potential to be integrated into the software stack of enterprise primary storage architectures.

Despite significant cost benefits of data reduction, enabling complex enterprise-grade data reduction in the software stack of all-flash storage systems can adversely affect the performance of write-intensive applications as depicted in Fig. 1b (further details in Sec. 2). This is very surprising as one expects, e.g., that by achieving 90% data reduction, particularly in write-intensive applications, the performance will improve by 10× as the disk subsystem will be 10× more available to serve other I/Os. However, this is not the case in our experiments and a *performance gap of over an order of magnitude (up to 35×)* exists between the real full-stack measurement and the expected performance of the all-flash storage stack in the presence of data reduction techniques.

Our analysis also reveals that data reduction induces complex data and metadata I/O patterns so that the I/O patterns observed at the disk subsystem are often no longer similar to the I/O patterns originally received at the *host bus adapter* (HBA) (Fig. 1a (III)). This shifts the optimal configuration points of SSD RAID arrays to completely different and *unknown* points. In particular, since enterprise data reduction techniques (e.g., VDO) have a very high complexity with over 150K lines of code, designing and integrating enterprise-level optimizations to the data reduction source



(a) Hardware/software stack in enterprise storage systems in presence of data reduction module (I: Disk subsystem, II: Software stack, III: Host bus adapter)



(b) Performance gap of optimal Data Reduction (DR) architecture vs. measured IOPS in write-intensive applications (*w*: with, *wo*: without)

Fig. 1. All-flash storage systems: a) hardware/software stack and b) performance gap with respect to optimal data reduction architecture

code for enhancing either performance or reliability (e.g., minimizing DRAM usage for metadata management or mitigating data corruption in case of power failures) to reach the optimal points would be extremely difficult.

Unfortunately, existing studies do not provide sufficient insight into the performance gap between data reduction and SSD-based RAID disk subsystems and mostly target the reliability aspect rather than performance [18, 19, 41, 53]. A few works address how deduplication may degrade read performance and propose controlled data replication for more distributed, higher-performance read requests [15]. However, such existing works examine the deduplication impact on *only* read performance and evaluate it on simulators *rather than a real system*. This necessitates the need for an in-depth study of the effect of enterprise data reduction on underlying enterprise SSD disk subsystems and determining how to design an optimal enterprise stack in the presence of data reduction. Unfortunately, to the best of our knowledge, **neither** the aforementioned performance gap **nor** the detailed impact of data reduction on the performance of enterprise all-flash storage systems has been explored in the previous work.

In this paper, we aim to reduce the performance gap between best practices used in the state-of-the-art and the optimal data reduction in all-flash storage systems. To this end, as the **first contribution**, we conduct a comprehensive set of experiments to explore the performance per cost of various RAID configurations while enabling the data reduction module (here, VDO) on top of the disk subsystem (as depicted in Fig. 1a). Our experiments on an enterprise all-flash storage system show several *unexpected* results, which completely **contradict** the common best practices used in all-flash storage systems. For example, unlike for conventional best-practice configurations, when data reduction is enabled, **a)** RAID 10 does *no longer* provide significantly higher performance for random workloads compared to other RAID configurations; as such, more cost-effective RAID types can be used in all-flash storage systems; **b)** increasing the RAID *span* (the number of parallel arrays configured as one large array) does not help to enhance performance; **c)** the write-enabled cache of the RAID controller can significantly enhance the overall performance, while the opposite is true without data reduction; and **d)** reducing the *stripe* unit size (the unit of distributing data blocks on the RAID array) can significantly enhance the array performance. Such *novel* findings can help storage architects reduce the overall cost per terabyte by enabling data reduction while simultaneously enhancing performance by up to 2×.

As the **second contribution**, to further explore possible opportunities to fill the performance gap, we examine the I/O pattern at the storage entry point and compare it with the data blocks written to the disk subsystem. Our investigation in this part reveals further novel observations. When data reduction is enabled on top of the SSD array, **a)** more than 72% of I/O requests are written into a very small address range (less than 0.2% of the total storage capacity); **b)** at 90% data reduction, unlike our expectation to write 10× less to the disk subsystem, we observe a significantly (up to 2×) greater amount of data written to the SSD array compared to the baseline without data reduction; **c)** we observe significant read overheads (up to 1.3× of write overheads) imposed to the disk subsystem, which are used for chunk fingerprint lookups, duplicate content verification, and address mapping lookups; **d)** at a higher content locality, we observe higher locality of accesses to data blocks used for duplicate verification.

As the **third contribution**, using our analysis of I/O patterns, we propose an efficient I/O cache architecture using the Intel *OpenCAS* I/O caching scheme. In the proposed caching architecture, we reserve a small part of the main memory and configure it as a *read-only* I/O cache on the path from VDO to the disk subsystem. Our experiments on an enterprise storage system show that the proposed I/O cache can improve the performance of write-intensive random workloads up to 2×.

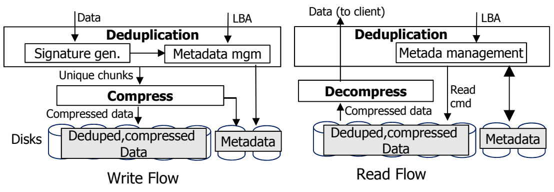
As the **fourth contribution**, motivated by different access patterns for metadata and data blocks, we extend our optimizations to the software stack and update the VDO source code to separate block map metadata from the data regions. By differentiating the metadata and data regions, we redirect and offload the block maps into a fast memory device (e.g., persistent memory DIMMs) and integrate it into the I/O block layer of the Linux operating system. Our experimental results show up to 3× additional performance improvement in write-intensive workloads. We further examine the effect of application-aware data persistence on full-stack performance. We show that providing *lazy persistence* and the freedom from conservative persistency to the upper application layers offers significantly less performance overhead on random access workloads and increases the SSD lifetime.

As the **fifth contribution**, we open-source all our cross-layer optimizations including a) the VDO modified source code, b) optimized RAID configurations, and c) the proposed I/O cache architecture to be used by industry and academia. Since all our cross-layer optimizations have been experimented on an enterprise all-flash storage system, it will be a valuable platform for future research directions on data reduction.

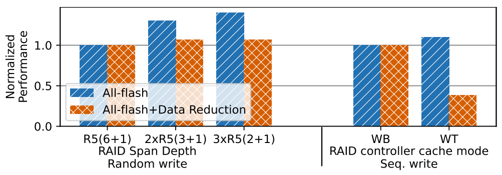
We organize the remainder of this paper as follows. We present data reduction basics followed by the effect of VDO on storage stack layers and its I/O overhead breakdown in Sec. 2. We detail our proposed evaluation methodology in Sec. 3. Next, we present our empirical best-practice exploration on all-flash storage systems with data reduction enabled in Sec. 4. We then propose our data/metadata placement architecture in Sec. 5. We further present the overall performance-cost evaluation of our proposed optimization techniques in Sec. 6. Then, we briefly review the related work in Sec. 7. We discuss the generality of our observations and possible design issues of our proposed architecture in Sec. 8. Finally, we conclude the paper in Sec. 9.

2 BACKGROUND AND MOTIVATION

This section first discusses the concept of different RAID configurations and then, based on VDO, how a typical data reduction module works, and lastly investigates the reasons for the significant performance gap between the optimal and the state-of-the-art VDO. Since we aim to enhance the end-to-end performance, we investigate *not only* VDO but also other layers in the storage stack, affected by VDO. We show how VDO changes the standard behavior of different RAID configurations, which drastically reduces the efficiency of the storage stack. Lastly, we analyze the



(a) Basic operational flow of data reduction. (LBA:Logical Block Address, PBA: Physical Block Address)



(b) Normalized IOPS and bandwidth of an all-flash storage system *with* and *without* VDO module

Fig. 2. Data reduction: a) basic operational flow and b) performance impact on all-flash storage systems

I/O footprint and working set size of different storage regions accessed by VDO to show the major sources of VDO overheads.

2.1 RAID Basics

A redundant array of independent disks (RAID) provides two functionalities for a group of disks: enhancing I/O performance and improving reliability in case of disk failures [49]. RAID improves I/O performance by dividing incoming data into smaller units called *stripe units* and distributing these units among multiple disks. The added reliability of RAID is achieved through either data replication (as in RAID 1) or generating and storing parities from a group of chunks (as in RAID 5 and RAID 6).

Parity-based RAID configurations are the most common RAIDs and follow a $N + K$ striping scheme. In this scheme, the incoming data is split into N data stripe units and K stripe parities are calculated and added to the original data. All stripe units (data and parities) are stored independently on an array of distinct $N + K$ storage devices. RAID 5 and RAID 6 are examples of parity-based RAID configurations. RAID 5 with one parity, denoted as RAID 5($N + 1$), can tolerate a single-disk failure while RAID 6 with two parities, denoted as RAID 6($N + 2$), can tolerate two disk failures.

A combination of different RAID types is possible for higher performance or reliability. A typical combination is RAID 0 (for only data striping) with a reliable configuration such as RAID 5 or RAID 1. For example, RAID 10 with 6 disks, configured as either RAID 10(3+3) or 3xRAID 1(1+1), distributes data into three spans (as in RAID 0 with 3 disks) while each span holds the data and its replicate (as in RAID 1 with 2 disks).

2.2 Data Reduction Operational Flow in VDO

When a *write request* is sent to the data reduction module, it first goes through the deduplication process (Fig. 2a). The data reduction module hashes the data block to calculate its signature (a.k.a., fingerprint). It then looks up the signature in the *Universal Deduplication Service* (UDS) metadata tables that hold the signatures and their corresponding physical addresses. If the signature is found, the block can be usually considered to be a duplicate. Nevertheless, many enterprise *primary storage* systems [12] and also VDO only store non-cryptographic *Murmur3* [21] hashes to maintain high hash rates and reduce the CPU pressure and thereby induced performance overheads on deduplication. To guarantee that the data is truly a duplicate and not a false alarm of a hash collision, VDO needs to fetch the corresponding data block from the SSDs and perform a byte-to-byte comparison to verify that both blocks are identical. In case of a duplicate, the data reduction module only updates the metadata including the mapping of the logical address used by the client to the physical address of the previously stored block with the same content. If the signature is not found in the UDS table, the data block is considered as being unique and must be stored on the disks. The data reduction module additionally compresses such non-duplicate blocks before storing them to further minimize the data footprint. VDO writes data blocks in a log-structured fashion and therefore transforms many write access patterns into sequential ones. VDO implements its data reduction service on top of raw block devices and handles all complexities of laying out data/metadata blocks and ensuring their consistency.

For a *read request*, the data reduction module typically has to do fewer operations than for write operations (Fig. 2a). It looks up the logical address of the request in the data reduction metadata (i.e., block mapping tables) and finds the physical location of the requested data. The module then reads the possibly compressed block from the SSDs, selects the compressed range, decompresses the block, and then forwards it to the client.

2.3 Effect of VDO on Other Storage Stack Layers

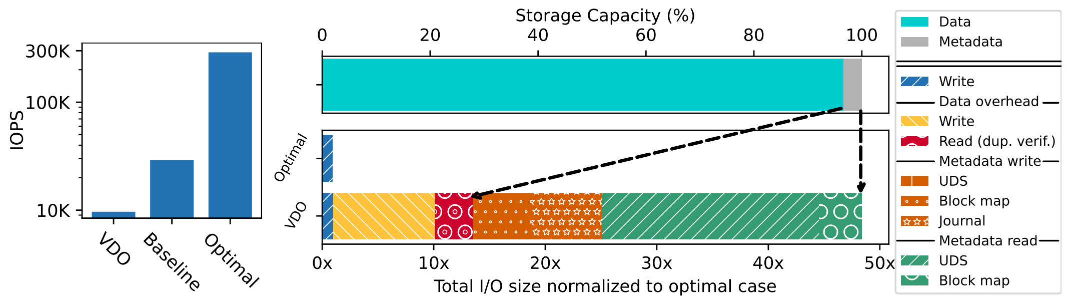
Computer system architects typically optimize the storage stack layers based on the input characteristics and the interaction between the layers. In many enterprise storage systems, such optimizations also depend on the application and performance, reliability, and cost requirements. Adding a complex component such as VDO can significantly modify the I/O patterns sent to other storage layers (in either software or hardware).

By conducting several experiments on an all-flash storage system, we observe that simply enabling VDO on top of an *optimized* stack can result in a *non-optimized* stack with significantly degraded performance. Fig. 2b shows the normalized performance of various RAID configurations *with* and *without* VDO. In the baseline all-flash, while running a random 8 KB workload with 75% data reduction potential, increasing the number of spans in a RAID 5 configuration enhances the parallelism and boosts the random write performance. However, this improvement becomes almost non-existent when VDO *deduplication* and *compression* are enabled in the software stack. Another example presented in Fig. 2b is following the industry recommendation for SSD arrays [42] to remove the RAID controller cache from the critical path while servicing sequential 4 MB write requests, i.e., changing write-back (WB) mode to write-through (WT). Following the recommended practice in the baseline increases the performance, since the cache has an almost 0% hit ratio and only adds flushing (and latency) overhead to the requests. However, when VDO is enabled on top of the *same* RAID controller, disabling the write cache degrades the performance by 62%. Such observations encouraged us to perform a thorough design space exploration and to investigate the full-stack storage performance in the presence of VDO.

2.4 VDO I/O Overhead Breakdown

Here we show that the VDO deduplication and compression stage is a major source of I/O amplification. For a write-intensive, randomly accessed workload with 90% data reduction potential, the I/O traffic submitted to the disk subsystem is commonly expected to be 10× lower than for the baseline without data reduction; however, the measured I/O traffic on a storage stack with VDO is even up to 5× higher than that of the baseline and thus 50× higher than optimally expected. This severely limits the performance of the all-flash storage to less than 10K I/O operations per second (IOPS), which was optimally expected to be 10× faster than the baseline of 30K IOPS (see Fig. 3a).

We observe that these significant overheads can be grouped into **a)** *write-only overheads*, i.e., data write/journaling overhead (18.7%), **b)** *read-only overheads* (47.1%) including UDS fingerprint lookups (40.1%) and duplicate verification (7%), and **c)** *mixed read-write overheads* (31.8%) including block map updates (18.7%) and recovery-journal updates (13.1%) (see Fig. 3b). *Data journaling* is used in VDO for a conservative approach to guaranteeing persistency, but it requires that client write requests are committed to the SSDs before being acknowledged, and afterward the data reduction is applied to the written content. VDO has already implemented several optimizations to



(a) Random Write Performance

(b) Storage Capacity and I/O Overhead Breakdown of VDO

Fig. 3. I/O overhead breakdown of VDO and its real performance for a random access write workload with 90% data reduction potential and its comparison with the expected optimal performance

improve journaling performance, which avoids data rewrites for non-compressible, non-duplicate content. Our observations nevertheless show that such immediate persistence, which is performed independently from the application requirements, induces a severe bottleneck for highly deduplicable or compressible content. UDS fingerprint lookups (used to *detect duplicates*) and duplicate verification (used to *ensure enterprise-grade reliability for deduplication*) constitute significant I/O overheads while they *only* access a very small SSD region, as shown in Fig. 3b. VDO has an optimized locality-aware layout for fingerprint storage and uses a two-level hierarchical in-memory compressed fingerprint cache to speed up such accesses. However, it still suffers from frequent SSD lookups for random write-intensive workloads. Our observation also shows that due to the VDO internal deduplication window (i.e., the recommended 1 TB in our setup of 10 TB physical capacity [51]), SSD lookups happen at less than 0.1% of the whole storage address range (10 GB region for 10 TB physical capacity). This motivates adding a read-only I/O cache to the storage stack and evaluating how it mitigates such read overheads.

Lastly, metadata updates in VDO mainly consist of block maps, chunk fingerprint mappings, and recovery journals (for block maps); among such updates, the significant overheads are due to block map updates and recovery journals. Considering the small footprint of block maps (five bytes per 4 KB) and the recovery journal (e.g., 128 MB in VDO), metadata updates happen to less than 1% of the storage address range. This motivates removing the metadata updates from the critical path by offloading block maps and recovery journals to a much faster memory device than SSDs (e.g., persistent memory DIMMs).

3 PROPOSED METHODOLOGY

Following the motivational results, designing an optimal data reduction architecture is difficult due to its (1) additional I/O accesses for data reduction metadata management, (2) highly complex mechanisms to ensure no data corruption in case of power failures, and (3) interaction with complex storage components such as hardware RAID controllers. These reasons have motivated us to propose multi-layer architectural optimizations through three layers from the disk subsystem up to the operating system and the data reduction software, as summarized in Fig. 4.

First, we explore the impact of disk subsystem configurations on the performance of an all-flash system *with* and *without* data reduction enabled and reveal five major findings on the efficient tuning of the storage stack with data reduction. We therefore explore major configurations of

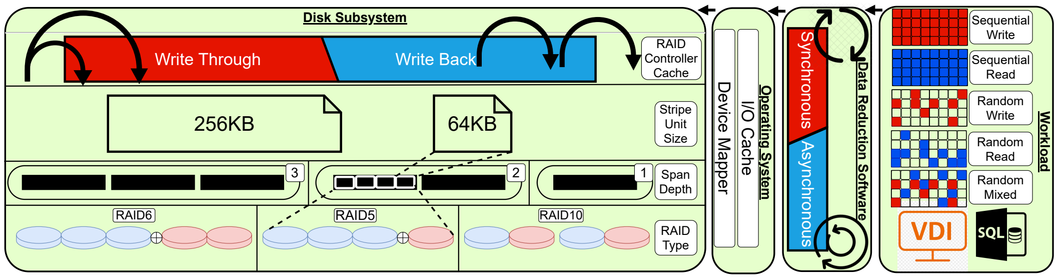


Fig. 4. Our proposed methodology of best-practice exploration on major parameters of an enterprise-grade storage stack and further storage architecture optimization to maximize full-stack performance

Table 1. Default Parameters used in Experiments (*RCC W-Policy*: RAID Controller Cache Write Policy)

RCC W-Policy	**Stripe Unit**	**Span Depth**	**VDO Sync.**	**Req. Size**	**Read/Write Ratio**
Write-Back (WB)	256 KB	R10:3, R5/6:1	Sync	8 KB	100/0, 0/100, 70/30

hardware RAID controllers including the RAID type, RAID span depth, RAID stripe unit size, and write cache policy. We also investigate the effect of the synchronous and asynchronous interaction between the data reduction software and the disk subsystem.

After exploring the impact of RAID configurations on an all-flash system with data reduction, we provide a detailed I/O overhead breakdown of the full stack. Then, we propose and evaluate a data/metadata placement architecture to minimize the remaining read/write overheads inherent to data reduction. We aim for a highly reliable design and easy adoption in real environments, thus we present a modular architecture, which stacks stable modules below the data reduction layer to reduce source code modifications down to only the required parts. Our proposed data/metadata placement architecture therefore consists of **(1)** an I/O caching architecture using OpenCAS that dynamically maps frequently required content to a faster storage device, and **(2)** a static metadata-data disaggregation architecture based on DM-Linear to map frequently accessed metadata as well as metadata journals into a region with fast memory access but *away* from the data regions.

System Setup: We have used a storage server with dual-socket Intel Xeon E5-2620 v4 CPUs, nine Samsung SM863a 1.92 TB SSDs, a MegaRAID 9361 RAID controller, RedHat VDO 6.1.3.23 as open-source enterprise data reduction module [50], and CentOS 7 upgraded to kernel version 5.4. VDO internally uses 4 KB data blocks, which is suitable for primary storage accesses. As an initial configuration for the performance exploration, we have configured VDO to use *synchronous write-mode* and a) 2 GB – 4 GB internal DRAM cache for block maps to cover 10% of the client address range, b) the recommended 1 GB DRAM cache for fingerprint mapping for physical capacities of less than 10 TB, and c) four threads instead of the default one for every major operation including compression and hashing. During the tests, deduplication and compression of VDO were enabled, while VDO only benefits from compression if a block can be compressed by more than 50%. We furthermore have used Oracle Vdbench to generate synthetic and realistic I/O workloads[1]. Throughout the experiments in the next sections, if not explicitly mentioned in our setup, we will follow the default parameters listed in Table 1.

Synthetic workloads: We have configured Vdbench to generate different access types (reads, writes, and mixed), different access patterns (sequential large 4 MB and random small 8 KB), and I/O

[1]Due to privacy reasons, no publicly available traces exist with both content (required for compression and deduplication) and I/O addresses. As such, we use a combination of synthetic and realistic I/O models, similar to the previous work.

data with the desired data reduction percentage potential ranging from 0% to 90%. The deduplication sets (i.e., the number of blocks with different content that are used to generate all the duplicate blocks) have been set to be around 5% of the total amount of duplicates as the default setting of Vdbench. The amount of data accessed during each test has been about 200 GB. For read tests, the I/O access range has been limited to 500 GB and the access range has been initialized by a sequential write of the dataset with a specified data reduction percentage. Such initialization is necessary to ensure read requests happen to already-allocated regions and the data reduction module returns valid block content.
Realistic I/O models: We have used publicly available I/O models of *SQL* databases and *virtual desktop infrastructures* (VDI) [43]. The SQL database has an average block size of 52 KB for reads and 32 KB for writes, 68.42% read accesses, 80% random accesses, and about 65% data reduction capability (50% from deduplication and an additional 15% from compression). The VDI workload has a fairly similar access distribution with a much higher data reduction potential of 97% (79% from deduplication and an additional 18% from compression). We have run the measurement phase for both tests to cover 100 GB I/O accesses. Before each test, we have initialized the first 500 GB I/O access range by sequentially writing a dataset with the same data reduction potential as the specified workload.

4 BEST-PRACTICE STORAGE ARCHITECTURE EXPLORATION

In this section, we explore the impact of architectural configuration decisions on all-flash storage with data reduction enabled and examine their performance/cost effects by comparing them with the baseline. We particularly examine the main parameters of the RAID-controller (RAID-type, span-depth, stripe-unit, and internal write cache) and the method of I/O dispatching (synchronous vs. asynchronous). Our explorations reveal insights to derive the required modifications on the storage stack configuration and show that well-known assumptions on architectural optimizations of all-flash systems without considering the data reduction are *no* longer valid.

4.1 Effect of RAID Type

Finding 1: *RAID 10 is considered the best-practice configuration for random access workloads. However, we observe that data reduction removes the large performance gap between different RAID types. Thus, the performance of RAID 6 becomes very similar to RAID 10 when enabling data reduction.*

System designers conventionally employ mirrored RAID (i.e., RAID 10) for random write workloads, as parity-based RAIDs require *Read-Modify-Write* (RMW) operations to update parities. The corresponding performance impact can be seen in Fig. 5, where RAID 6 and RAID 5 are 2.7× and 3.2× slower than RAID 10 in the baseline (BL) random write configuration, respectively. Data reduction, however, reduces this gap to less than 1.4× for data reduction ratios higher than 75%.

This performance gap becomes even smaller, nearly non-existing, for mixed read-write workloads, as the performance of random reads usually does *not* heavily depend on RAID types. Fig. 5 shows for a mixed 70/30 read/write workload that RAID 10 is still 2× faster than RAID 6 in the baseline while enabling data reduction completely eliminates this performance gap between RAID 10 and RAID 5/RAID 6 by **a)** increasing CPU and I/O complexity and therefore reducing overall write IOPS and **b)** transforming the majority of RMW accesses to log-structured sequential accesses.

For applications such as VDI with mixed read-write workloads and data reduction ratios beyond 75%, RAID 6 with data reduction provides over 1.3× greater effective capacity than RAID 10 in the same setting while delivering comparable performance, making it a very efficient solution in terms of both cost and performance. This efficiency can even be increased for RAID 6 settings with more data disks and high data reduction. For example, with 75% data reduction, RAID 6 with VDO

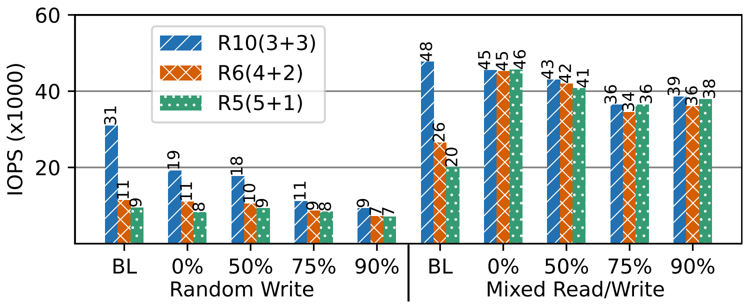


Fig. 5. Impact of RAID type on performance. BL in all figures represents the baseline without VDO, while percentage numbers depict the possible data reduction for VDO-enabled configurations.

provides 5.2× higher effective capacity compared to the baseline (no data reduction) at RAID 10, with only 30% lower performance (before applying our optimizations).

Implication 1: *Enterprise-grade data reduction enables using more cost-effective RAID types (e.g., RAID 5 instead of RAID 10) with similar performance, so that only the required reliability level dictates the RAID type and not the performance.*

4.2 Effect of RAID Span Depth

Finding 2: *For parity-based RAID configurations, increasing the number of spans (from one to three) on the baseline leads to up to 30% additional SSD cost, but boosts random write performance by over 2×. However, with data reduction enabled and a workload with over 50% data reduction potential, the added spans either do not increase performance or the performance increase is proportional to the added SSD cost.*

Fig. 6 shows the impact of span depth on random small-block access workloads and realistic workloads (SQL database and VDI), respectively. For parity-based RAID (e.g., RAID 5) and random write workloads, increasing the number of spans from one to three adds up to 30% additional SSD cost, but boosts performance by over 2× in the baseline. Such performance improvement is due to fewer RMWs and increased parallel accesses in each span independently. Enabling data reduction, however, converts most of the random data writes to sequential accesses. Hence, very few RMWs are observed even in a single span and additional overheads of data reduction operations (e.g., metadata updates) also limit the performance benefits of multiple spans. Therefore, the span depth is no longer beneficial. Note that span depth mainly affects random small accesses and has no

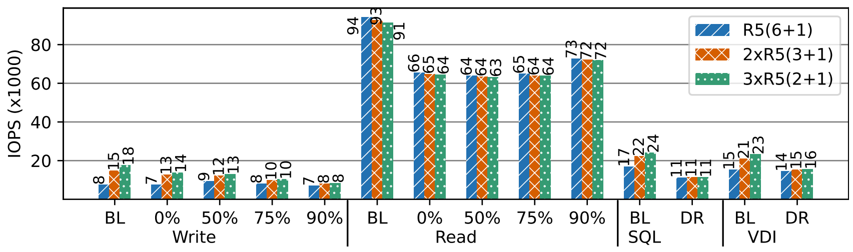


Fig. 6. Impact of RAID span depth on the performance of random write-only and random read-only I/O workloads and two realistic models, SQL database and VDI (*BL*: baseline, *DR*: data reduction enabled)

benefit on sequential large-block workloads both in the baseline and the data reduction enabled system, thus sequential access workloads are not shown in this experiment.

Implication 2: *To achieve the best trade-off between the array cost and performance in random workloads, the most efficient configuration for the baseline is to maximize the number of spans while a single span offers the best performance per cost for a system with data reduction.*

4.3 Effect of RAID Controller Cache

Finding 3: *On all-flash arrays, not only the small write cache of the RAID controller has no benefit for the baseline but it also causes 70% performance degradation in the case of RAID 10. However, the trend is almost reversed when data reduction is enabled, where the write cache can improve performance by up to 3× in the case of RAID 5.*

Storage vendors often recommend turning off the RAID-controller write-cache in all-flash systems for random access workloads, since random (uniform) writes do not have enough locality to take advantage of cache memory [8]. The missing locality can lead to cache thrashing so that the write cache can become a performance bottleneck. This can also be seen for the baseline in Fig. 7a, where enabling caching results in up to 70% performance degradation for RAID 10.

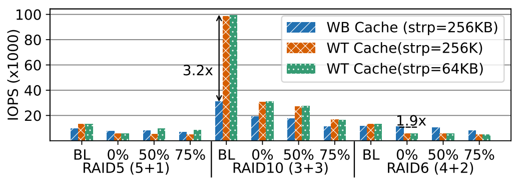
On the contrary, data reduction leads to many metadata accesses (log-structured 32 KB for chunk signature accesses) and also reorders data writes to become almost sequential at a small granularity of around 4 KB. In parity-based RAIDs, such small I/O accesses result in many RMWs updating parities. Therefore, using even a small RAID controller cache (1GB in our case), such data writes and extra metadata pages may coalesce and send larger and more efficient blocks to SSDs. By checking the number of reads and writes received by each SSD (using the SMART [2] monitoring tools [32, 56]), we observe that the RAID controller cache reduces the additional read overheads (by up to 2× in random write workloads with RAID5) and thus improves data distribution across SSDs, resulting in up to 1.4× performance boost. For the RAID 10 setup, the data reduction enabled system may provide slightly higher performance with WT cache, but due to additional data reduction overheads, its benefit is very limited.

Sequential large writes with data reduction lead to a similar performance trend compared to random workloads due to the presence of small-block accesses caused by metadata lookups/updates, and VDO dispatching data blocks at limited granularity (Fig. 7b). Such similarity becomes more visible on parity-based RAIDs, as data reduction leads to small-sized metadata writes and can cause many RMWs. The write-back cache of the RAID controller minimizes these overheads such as the RMWs (up to 50×, based on measured SSD SMART statistics), and thus we observe a performance boost of up to 3.2×.

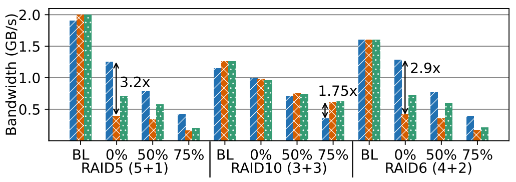
Real workload models of SQL database and VDI also exhibit similar behavior to the synthetic write workloads (Fig. 7c). The WT mode of the RAID controller cache improves the performance of the baseline (no data reduction), but such improvement is small due to the read-intensiveness of SQL and VDI, and naturally, there is little dependence on the RAID controller write cache. However, WB cache is still significantly effective for data reduction performance and provides 2× higher performance for parity-based RAIDs (the opposite of the baseline). Overall, data reduction heavily relies on the write-cache, while it does not provide much help for the baseline in all-flash storage systems.

Implication 3: *To achieve the best performance, the RAID controller write cache should be disabled for the baseline configuration. On the contrary, with data reduction added and parity-based RAIDs used in the storage stack, we achieve a significant performance benefit from an enabled WB cache.*

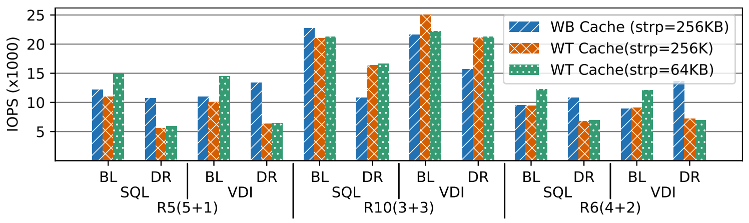
[2]Self-Monitoring, Analysis, and Reporting Technology



(a) Random Writes



(b) Sequential Writes



(c) Real Workload Model

Fig. 7. Impact of RAID controller cache policy and stripe unit size on the performance of different workloads for write-through (WT) and write-back (WB) caches (*BL*: Baseline, *DR*: Data Reduction enabled).

4.4 Effect of RAID Stripe Unit Size

Finding 4: *Reducing the stripe unit size from the default 256 KB to 64 KB has less than 1% performance impact on the baseline. However, with data reduction and no write cache, up to 2× speedup is achieved by reducing the stripe unit size. With write-cache and data reduction enabled, the performance impact of the stripe unit size becomes less than 10%.*

Fig. 7 shows the effect of the stripe unit size on the performance of random and sequential workloads. Sequential large (4 MB) writes in the baseline do not depend on the stripe size, as the stripe size has almost no effect when the request size is larger than the stripe. Random small (8 KB) writes also do not show a dependence on the stripe size, due to the limited locality of the workload. In the case of all-flash storage systems with data reduction, however, even 4 MB data writes are submitted to the underlying RAID controller as small blocks with an average size of less than 30 KB. In the case of parity-based RAID configurations, this granularity of I/O accesses comes with extra RMWs for parity updates, resulting in performance degradation. With a smaller stripe unit size, the chance of merging multiple writes into a single stripe and avoiding RMWs increases significantly. Similarly, in case of random workloads, part of the I/O accesses are in the form of log-structured sequential writes of data and metadata. Therefore, both small stripe unit size and write cache boost the performance.

Implication 4: *For the baseline, keeping the default stripe unit size of 256 KB is sufficient for both large sequential writes and small random accesses. However, when data reduction is enabled, setting the stripe unit size to the minimum (e.g., 64 KB) is required for the best performance (if write cache is not available).*

4.5 Effect of VDO Sync/Async Modes

Finding 5: *VDO asynchronous write mode removes up to 2.6× of additional writes compared to the synchronous mode for sequential workloads but shows only a little improvement in case of random I/O.*

Depending on the application requirements, the write path from the data reduction software to the RAID subsystem has two modes: *synchronous* (sync mode) and *asynchronous* (async mode). The

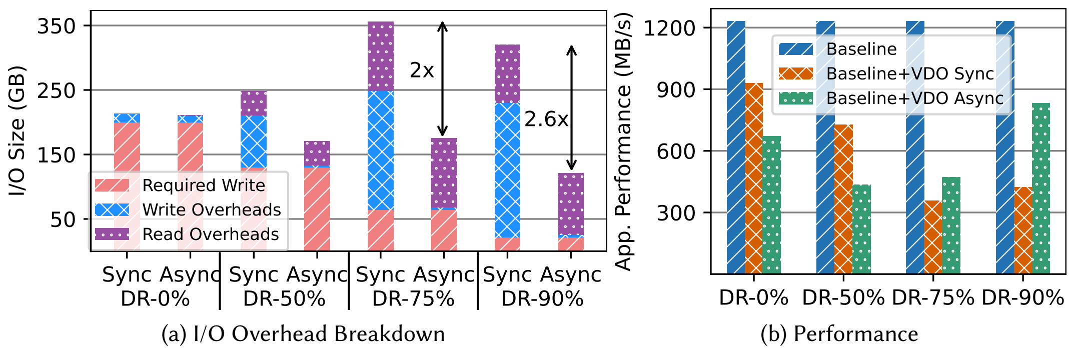


Fig. 8. Comparison of VDO synchronous and asynchronous mode for sequential write workloads on RAID 10(3+3). The asynchronous mode shows (a) significantly reduced I/O overheads and (b) major performance gains for high reduction ratios (≥ 75%).

sync mode guarantees strong persistence regardless of application requirements while the async mode relies on the application to request persistence guarantees. For the sync mode, a data block is written first to the underlying storage device, the write request is acknowledged to the host, and then the data reduction core operations (deduplication and compression) start. If the data block is non-duplicate *but* compressible, it will be rewritten to SSDs at a new location; if it turns out to be a duplicate, it will be reclaimed later. In the async mode, the write requests are acknowledged to the application as soon as the requests are written to the DRAM, and then the data reduction operation starts. Moving the control of guaranteeing data consistency to the application through explicit flush commands enables the data reduction software to reduce the amount of I/O writes to SSDs, and thus a higher performance is expected.

Fig. 8 shows that the VDO async mode eliminates almost all write overheads for sequential workloads, decreasing the total amount of I/O by up to 2.6× and boosting the write bandwidth by 2×. The difference between VDO-async and VDO-sync writes is the amount of duplicate content or non-duplicate, but compressible content. As a result, when the data reduction potential of a workload increases, the async mode becomes much more efficient than the sync mode for sequential workloads, causing fewer writes to SSDs, thus improving the SSD lifetime and the write performance. Eventually, only read overheads remain, which are mainly due to reading data blocks with the same chunk fingerprint to perform the duplicate verification. Note that for workloads with data reduction potential of less than 50%, the async mode does not improve the performance, due to minimal duplicate write overheads in such cases. Current implementation of VDO also results in larger I/O batches submitted to SSDs in sync mode compared to async mode when there is 0% to 50% data reduction.

For random access workloads, async mode also significantly reduces the write overheads, but unlike the initial expectation, such overheads only form up to 25% of the total overheads (Fig. 9). In such workloads, the data reduction module must update a logical-to-physical address mapping for every 4 KB access. However, unlike the corresponding updates for sequential workloads, such updates cannot be completely contiguous, which results in many random reads and writes. In addition, part of the read overheads is caused by duplicate verification. Therefore, a decent portion of write overheads can be avoided through VDO async mode and SSD lifetime can also be boosted, but the overall performance impact of switching between the *sync* and *async* mode is very limited.

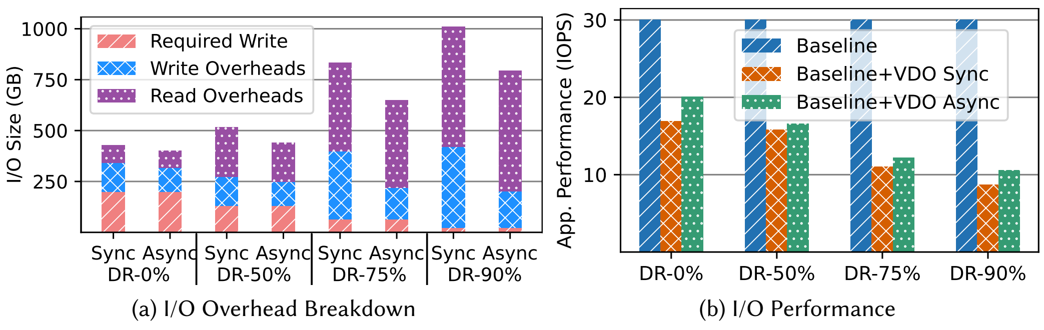


Fig. 9. Comparison of VDO synchronous and asynchronous mode for random write workloads on RAID10(3+3). Minor impact of asynchronous mode in data reduction on (a) I/O overheads and (b) performance.

Implication 5: *While providing strong persistence regardless of the application type in the storage system has less performance overhead on random access workloads, providing the freedom to the upper application layer to request for strong persistence through flush commands provides a high potential for performance boost and SSD lifetime improvement for write-intensive workloads, especially for high data-reduction sequential workloads.*

5 DATA/METADATA PLACEMENT ARCHITECTURE

Earlier, we showed how to configure the RAID controller and the write-path sync mode in the storage stack to achieve the highest performance *with* data reduction enabled. We also observed that even after a proper configuration, a majority of the additional read/write I/O overheads remain for both random access and sequential access workloads. To overcome these remaining overheads, we propose two architectural optimizations: **a)** *I/O caching with read-only policy for UDS lookup (fingerprint read) and duplicate verification (data read)* acceleration and **b)** *the disaggregation of block maps from the data* for offloading them to a faster, and more suitable memory. Caching highly accessed content, or offloading data reduction metadata to a faster device are not new ideas and are well studied in the previous work [2, 3, 54]; however, our *main contributions for introducing these two architectural optimizations* are **a)** to elaborate and clarify how such ideas affect the performance of a real all-flash storage stack and **b)** to provide effective guidance on the implementation of such ideas using open-source, enterprise-grade components including *RedHat VDO* as the data reduction layer, *OpenCAS* as the caching layer, and *DM-Linear* as support for disaggregation of data/metadata.

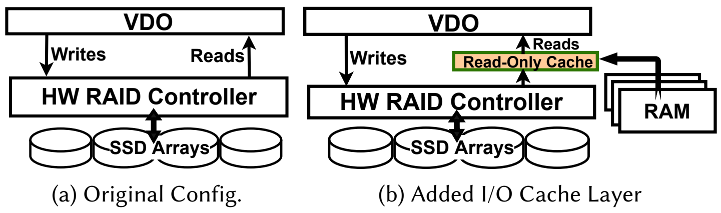


Fig. 10. Data/metadata placement optimization: a read-only I/O cache to accelerate UDS fingerprint lookup and duplicate verification data reads

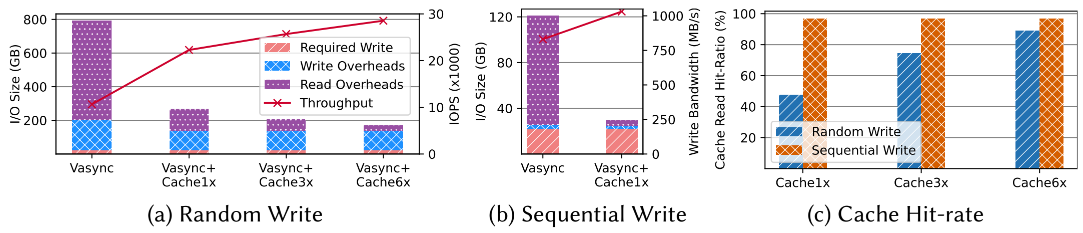


Fig. 11. Effect of the proposed read-only cache for UDS fingerprint lookups and duplicate data verification on random/sequential write workloads with 90% data reduction [measurements on RAID 10(3+3)]

5.1 I/O Cache Architecture

Architectural idea. To reduce the read overheads of the UDS fingerprint lookups and duplicate verification process, we add a read-only I/O cache layer under the data reduction layer to capture frequently accessed fingerprints and data blocks (Fig. 10). We recommend a fast, possibly volatile memory type (e.g., DRAM) as the caching device due to its high speed, little required cache size, and *no* requirement for persistence. Our recommendation relies on two facts. First, DRAM is at least two orders of magnitude faster than SSD arrays, thus we expect a high potential for performance boost. Second, although the fingerprints and related metadata of data reduction may be huge (e.g., 850 GB for 100 TB physical capacity), VDO considers that some localities typically exist for deduplication (i.e., duplicate content appears within a limited window of written content); thus chunk fingerprint lookups happen only to a small address range. For example, following the RedHat VDO recommendation, using VDO compressed indexing format (i.e., sparse indexing) and deduplication window of 30 TB for 100 TB physical capacity, only 255 GB address range (out of 850 GB) are frequently accessed for fingerprints [51]. Third, the I/O cache is below the data reduction logical-to-physical address mapping, which means for multiple logical addresses that point to the same data content, only a single data copy would be required in the cache for cache hits. Therefore, the cache access hits follow content locality (not just address locality). Such content-based caching improves the efficiency of I/O caching further even at small sizes [33].

Implementation note. We employ OpenCAS [24, 25] as our I/O cache module in the Linux device-mapper layer and set its caching mode to *read-only* (i.e., write-around in OpenCAS terminology) to avoid cache pollution imposed by write requests. We also set the cache line size to 4 KB to match the granularity of the deduplication chunking and keep other configuration values the same as the default of OpenCAS. To analyze the effect of our read caching more accurately, we estimate the total size of compressed hot duplicate content (from Vdbench configuration) and the size of stored chunk fingerprints and set the cache size accordingly. With the above considerations, we use a cache size of 4.5 GB as the default for 90% data reduction test cases (marked as *Cache1x* in Fig. 11). When we test larger caches, we explicitly mention (e.g., *Cache3x* for a 3× larger cache in Fig. 11).

Finding 6. *Adding a small-sized read-only cache to capture UDS fingerprint lookups and duplicate verification reads of the data reduction layer eliminates the read overheads by up to 95% (in sequential workloads) and 78% in random workloads (67% of total read-write overheads). It also speeds up the write IOPS up to 1.24× (in sequential workloads) and 2× (in random workloads) for enterprise-grade data reduction on an all-flash storage system.*

In Fig. 11a, we show the amount of I/O overheads imposed by VDO data reduction in practice, in comparison with the optimal data reduction (labeled as *Required Write*). The results are reported for

90% data reduction which is feasible in popular VM-based workloads such as VDI. Such high data reduction workloads are of more interest to investigate, as they show more potential for storage cost savings. In this experiment, the amount of I/O that the workload generates is 200 GB (in the case of both random write and sequential write workloads), which translates to 20 GB of physical writes on the disk subsystem by considering 90% data reduction.

As shown in Fig. 11a and Fig. 11b, random-write and sequential-write workloads show significantly different data reduction overhead and performance behavior when I/O caching is integrated into the storage stack. With a cache that is large enough to hold all frequently accessed fingerprints and data content (*Cache1x*), the read overhead is reduced from 590 GB down to 129 GB in the random access workload and from 90 GB down to 5 GB (i.e., almost eliminated) in the sequential workload. The small I/O cache provides 96% cache hit-rate, removing all read overheads for the sequential workload that come from the duplicate verification reads and little UDS fingerprint overheads. In the random-write workload, however, an equally-sized cache provides 47% hit-rate, mitigating the read overheads and providing 2× higher performance. But removing all overheads and reaching over 90% cache hit-rate demands 6× larger cache size (Cache6X), which only increases the performance by 30% (as shown in Fig. 11a and Fig. 11c). These results show that the random-write workloads suffer from (1) significant UDS lookup overheads (430 GB vs. 20 GB in sequential workload) and (2) randomness and poor locality of block map lookups (random addresses vs. sequential addresses). While for the first, the I/O caching captures most of the lookups and removes them, the poor locality of block map lookups causes I/O cache pollution and motivates disaggregating them from other metadata/data accesses (our next optimization).

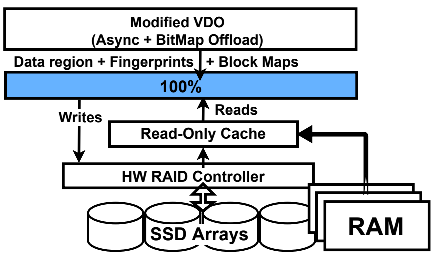
Finding Implication 6. *Even a small read-only DRAM cache can significantly decrease the read overheads of VDO and provide a performance boost, especially on random write workloads. In the case of workloads with a random write nature, by further increasing the cache size, the read overhead asymptotically converges to a specific value, motivating further improvements - apart from caching - in managing other metadata types (e.g., block maps).*

5.2 Metadata and Data Disaggregation Architecture

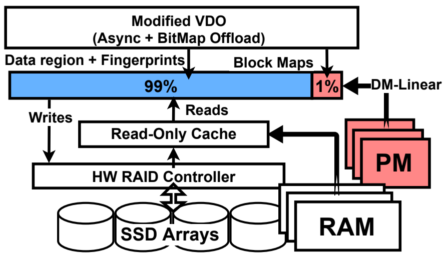
Architectural idea. After adding a small-sized read-only I/O cache to the stack, *over 10× I/O overheads* compared to the required expected I/Os still exist for the data reduction applied to random access workloads. Almost all of these remaining overheads are due to block-map updates and recovery-journal metadata writes (as highlighted in Section 2). The important insight is that such updates happen to a very small physical region of less than 1% of the storage space.

To address such overheads, we propose disaggregating the updates of block maps and the recovery-journal region from the data regions and offloading them to a small-sized low latency, high bandwidth, high endurance memory such as persistent memory DIMMs or battery-backed DRAM (Fig. 12). Note that such updates are more suitable for byte-addressable memories, as each 4 KB chunk requires modifying an average of five bytes. Such small-size updates are usually much smaller than SSD page size (4 KB - 8 KB), and even merging lookup/update requests into one large block may reduce the SSD performance and lifetime as well.

Implementation Note. VDO stores data blocks in contiguous physical regions of configurable 2 - 32 GB size, called *slabs* (Fig. 13). The original design of VDO follows the common log-structured design of data reduction solutions that sequentially writes the received 4 KB blocks regardless of the application address pattern and stores block maps to keep track of address mappings. VDO stores the block maps alongside data blocks in the same slabs. The motivation behind such binding of data and block maps to the same slab might have been to easily scale capacity by simply increasing the number of slabs. However, due to different behavior and granularity of accesses for block maps

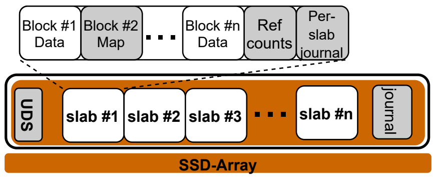


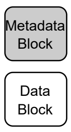
(a) Original Configuration + I/O Cache



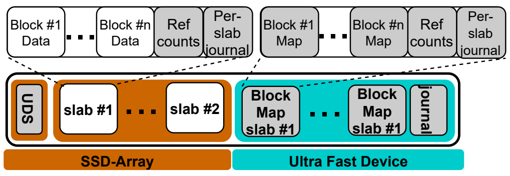
(b) BlockMap Offloading into Persistent Memory (PM)

Fig. 12. Proposed optimization for block map/recovery journal accesses





(a) Normal VDO Storage Layout



(b) After Separating Block Maps

Fig. 13. Data and metadata layout of data reduction module a) before and b) after our second optimization

and data blocks, and also due to significant overheads that metadata accesses impose on the system, we implement a new architecture to disaggregate and offload them to a different memory type.

To provide the block-map and data disaggregation while minimizing the source code modification, we have modified the VDO source code to use a predefined number of slabs at the end of the physical address space for block maps and use the remaining slabs for data blocks. The rationale of using the last set of slabs for metadata is that this region precedes the VDO recovery journal region, so offloading block maps and the small (about 256 MB) but also frequently accessed journal region to a different device is simplified. The number of slabs for block maps follows the logical size of a VDO volume. The choice of the logical volume size, which indirectly depends on the expected data reduction ratio, can be partly over-provisioned at the time of starting the data reduction service on a volume of data. For example, for a volume used by VDI workloads, with 10 TB physical raw space, 30 TB - 100 TB logical address space can be expected (i.e., 3×-10× reduction ratio). Each 4 KB block requires about five bytes to store its address mapping and each slab is 2 GB - 32 GB; as such, the number of slabs can be calculated accordingly.

To offload the disaggregated block maps and recovery journal to a fast memory, we use the DM-Linear module stacked under the VDO data reduction module (Fig. 14). DM-Linear is an open-source device mapper [64] that creates a composite logical device on top of other block devices and allows to redirect I/O requests at arbitrary address ranges of a volume to specific devices. We set up the DM-Linear address space such that it redirects almost the first 99% of physical address space (i.e., data block requests) to the SSDs (including the underlying read-only I/O cache) and redirects the requests for the last 1% of the physical address space (i.e., block maps and recovery journals) to the fast persistent memory. Note that it would be possible to modify the VDO source code more heavily to directly redirect I/O requests, however, by using the well-known and stable module (e.g.,

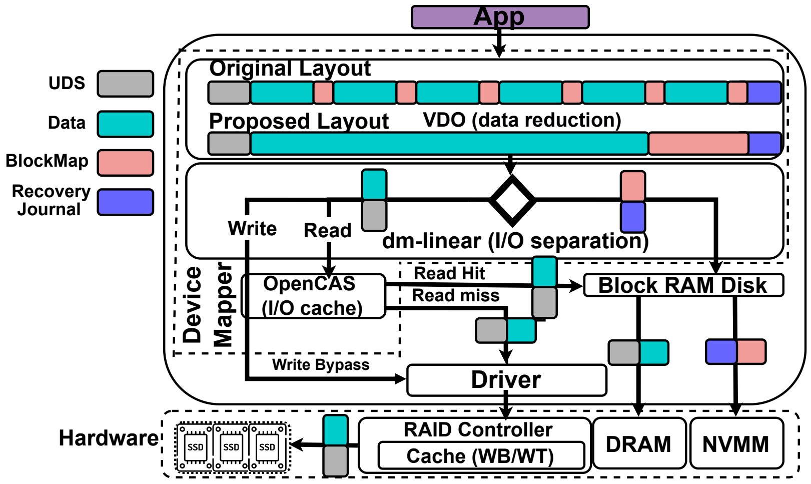


Fig. 14. Overall Proposed Architecture

DM-Linear), fewer changes to the existing data reduction framework would be required, and thus, we expect higher reliability and broader adaptation in existing storage stacks.

Finding 7. *Offloading the small-sized block maps and recovery-journal updates of VDO to a fast memory results in removing the remaining I/O overheads by up to 90% and achieving an additional 3× IOPS speedup over our first optimization (i.e., I/O caching) for write-intensive workloads while maintaining the same data reduction ratio.*

Fig. 15 shows the significant performance improvement for a write-only workload with high data reduction potential (i.e., 90%) when offloading the block-map and recovery-journal updates to fast persistent memory. After adding a small-sized read-only I/O cache in the VDO storage stack, and achieving a 2× performance improvement, a large portion of read/write overheads still remain. By offloading the block maps and the recovery journal, such overheads are almost removed (reducing from 246 GB down to 20 GB). By freeing more SSD bandwidth for data operations, avoiding the I/O cache pollution due to block-map accesses contending with UDS/data verification lookups, and also exploiting the low latency of the fast persistent memory, we observe an additional 3× IOPS speedup for write workloads. Note that block-map offloading and recovery-journal overheads are not dependent on the data reduction ratio, but rather on the address pattern of the workload. Sequential workloads have almost no overhead for block-map accesses (Fig. 11b), but common primary workloads with random accesses such as databases significantly benefit from this optimization.

Finding Implication 7. *Offloading the block maps and the recovery journal updates of VDO to persistent memory provides a significant boost on the performance of the all-flash system for random access workloads at only a marginal added cost.*

6 OVERALL PERFORMANCE-COST EVALUATION

In this section, we first present the overall performance speedup and I/O traffic reduction that our proposed architectural optimization provides over the partially tuned data reduction with VDO sync mode as well as over the baseline AFS with no data reduction. We then discuss the performance per cost of the system and show that our optimizations can provide up to 57× performance/cost improvement over the baseline for write workloads with high data reduction potential.

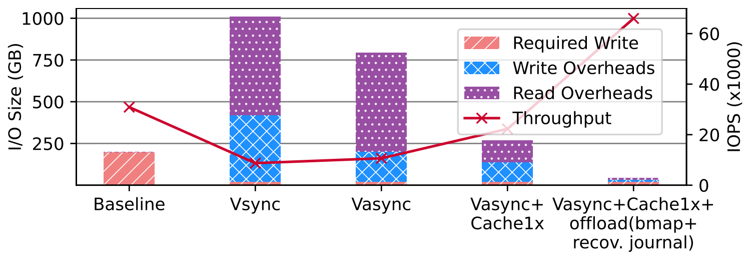


Fig. 15. Performance improvement of random access write workload by offloading block maps to a fast memory device (in our setup: DRAM with a battery-backed server). [measurements on RAID 10(3+3)]

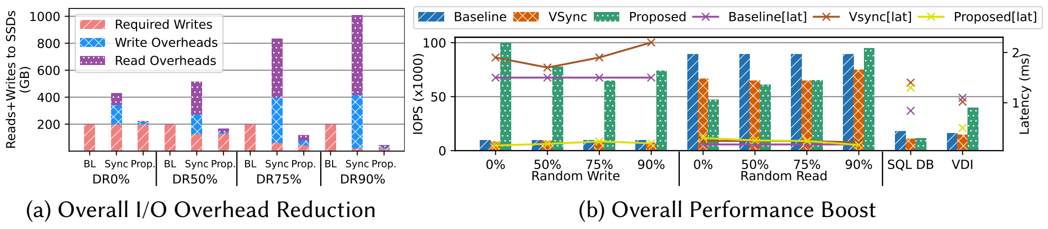


Fig. 16. The impact of our proposed architectural improvements and software optimizations on (a) overall I/O overhead reduction in random write workloads and (b) overall performance improvement for random read, write and mixed realistic workload models. [Results for RAID 5(5+1)]

6.1 Performance

Our proposed optimizations in the architecture of a storage system with data reduction including the proper data/metadata placement and modification of the VDO software module eliminate up to 98% of I/O overheads for data reduction (Fig. 16a). Overall, our optimizations provide *an order of magnitude improvement on IOPS and latency* over VDO sync mode (VSync) and the baseline (with no data reduction benefit) for write-intensive applications and up to 2.5× performance improvement for read-intensive workloads with high data reduction potential (Fig. 16b). Hence, employing our proposed optimizations in the storage stack with the VDO module provides up to 90% data reduction (which translates to 10× reduction in main storage media cost), alongside 2-10× improvement on IOPS and latency, depending on the read/write intensity of the workloads.

6.2 Final Note on Performance Cost

As final results, we compare the performance/cost of four schemes: **a)** baseline all-flash, **b)** all-flash with partially-tuned data reduction (Vsync), **c)** all-flash with all of our proposed architectural optimizations, and **d)** an optimal (i.e., close to ideal) all-flash configuration. We calculate the cost of these systems for three popular RAID configurations (RAID 10, RAID 5, and RAID 6) for a random write workload with 90% data reduction and also consider the costs of SSDs, CPU usage for data reduction, DRAM usage for the proposed caching (frequently accessed fingerprints and hot data), and NVDIMM usage for block-map offloading. We use the component prices following major online shops (DRAM with 6.25 \$/GB, NVDIMM [Intel Persistent Memory 200 series][3] with 7.4 \$/GB, CPU [Intel E5-2620v4] with 47.6 \$/core, SSD [Samsung SM863a] with 276 \$/TB) in our calculations.

[3]We used RAMDisk (and assume battery-backed server) in our testing, thus NVDIMM cost numbers are only for reference.

Our proposed full-stack optimizations provide up to 12.5× performance improvement over a partially-tuned data reduction (Vsync) at only 20% higher cost, and 57× performance/cost ratio improvement over the baseline all-flash for write workloads with high data reduction potential, thus enabling building an enterprise system with almost optimal performance/cost value (Fig. 17). Our optimizations achieve a performance level that is only 10%-25% lower than an almost ideal implementation of data reduction on parity-based RAIDs. For RAID 10, the performance gap between our proposed architecture and the optimal implementation is about 3.5×, which appears as a CPU bottleneck of the current implementation of our storage stack, e.g., due to thread contention of running over 40 threads for VDO, OpenCAS, and Vdbench on our 16-physical core/32-logical core system. We expect that by using a more recent generation of CPUs that have more CPU cores at decent prices, the performance gap with the ideal implementation on RAID 10 can be further mitigated.

7 RELATED WORK

Previous work on data reduction can be classified into three groups: (a) software-based data reduction solutions, (b) analysis of data reduction interaction with other system components, and (c) hardware-based data reduction solutions. In this section, we discuss existing work in each category and elaborate on how they differ from our proposed architecture.

7.1 Data Reduction Software

Many existing studies on data reduction focus on HDD arrays [14, 16, 58, 74]. Most of these studies propose optimizations to minimize fingerprint lookup accesses to HDDs during duplicate chunk detection to improve the write throughput of backup workloads. For example, the use of Bloom filters [74], improving the policy of caching fingerprints in DRAM based on the spatial or temporal occurrence of duplicates [38, 70], sorting fingerprints on the client-side to cooperatively process them interval-based at the server-side [29], using an SSD for metadata placement [14, 45], or sampling [20] have been proposed in the previous work. Others such as P-Dedup [71] propose software-pipelining to exploit multi-core CPUs and provide higher throughput.

For primary storage deduplication systems [14, 16, 44, 58, 67], latency is typically more critical than in backup scenarios. Thus, traditionally they delay data reduction to system idle time [16] to serve requests with low latency. While inline deduplication solutions can identify duplicates before they are sent to the disks, they need to avoid the fingerprint disk bottleneck. Thus, they often

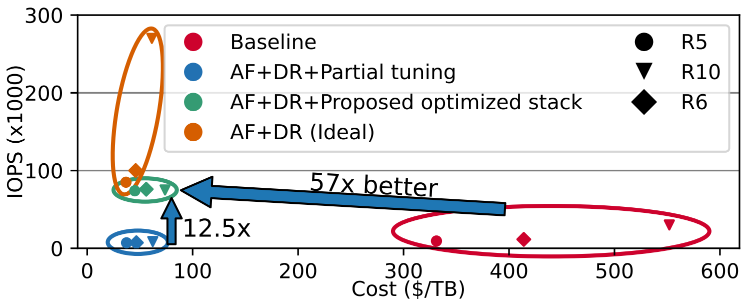


Fig. 17. Cost-Performance comparison of our proposed optimized storage stack architecture for data reduction on all-flash storage using a random write workload with 90% DR potential compared to no data reduction system and optimal solution. Note: Tests are for three widely-used configurations, i.e., RAID 5(5+1), RAID 10(3+3), and RAID 6(4+2).

reduce HDD lookups by **a)** storing the fingerprint index on SSDs [45], **b)** using memory caching strategies based on assumptions on the spatial locality of duplicates [67], **c)** using a fixed-sized cache to leverage the temporal locality of duplicates and detect them when they appear within a certain window [58], and **d)** combine both, a memory-cached inline-stage and a post-processing stage targeting an SSD-located fingerprint index on a cache miss [68]. To reduce index size and improve compression opportunities with less duplicate detection, or avoid data fragmentation, some solutions increase the chunking granularity or use thresholds for the minimum number of consecutive duplicate chunks [16, 58, 68]. ThinDedup proposes grouping the deduplication metadata with compressed data blocks to write them as a single large block and reduce metadata small-write overheads on an SSD [46]. For a fast metadata lookup, however, their scheme requires re-writing the metadata to a separate metadata table on the SSD during the system idle time, and if little idle time is available, double metadata writes cause performance drops. Some existing studies focus on fast phase change memory (e.g., NVDIMMs) as the back-end storage. For example, Zuo et. al [76] propose a lightweight deduplication for encrypted data on NVDIMMs, and NV-Dedup [65] takes advantage of NVDIMM byte-addressability to reduce the deduplication metadata update overheads on these devices.

In this paper, we focused on all-flash storage systems, which have the potential for large capacities in *Storage Area Networks* (SANs) (unlike NVDIMM-based systems), do not suffer from random accesses (unlike HDDs), and consider primary storage workloads that often serve random accesses rather than sequential workloads of backup environments. Our employed data reduction software, RedHat VDO, also employs a variety of optimizations that have been proposed in existing studies. Such optimizations are, for instance, a multi-threaded, pipelined design, along with an efficient chunk fingerprint layout. These optimizations of the original VDO have already removed some of the overheads that existing works have tried to address.

Several studies have proposed open-source software solutions, which provide data reduction [47, 50, 57, 60, 63]. The corresponding evaluations, however, have *not* included the performance interactions of the data reduction software with other storage components. In addition, some of these platforms are implemented at the file system level [47, 63], which is less suitable for our target environment of a flexible SAN storage environment combined with any desired file system. This also applies to deduplication solutions that target individual SSDs by implementing data reduction directly in the flash translation layer [10, 69, 72] and thus do *not* provide global reduction spanning across multiple devices. Instead, we focus on solutions that allow flexible integration of data reduction techniques into existing storage stacks with any file system, which are fairly independent of the Linux distribution by providing a stackable virtual volume abstraction in device-mapper layer using VDO [50].

7.2 Interaction of Storage Components

A limited number of studies evaluate the effect of combining data reduction with some storage components. For example, combining deduplication with I/O caching is an effective way to increase the cache hit rate [33]. Introducing flash caching layers in the deduplication stack has been shown to reduce read and write accesses to HDD-based primary storage solutions while simultaneously lowering the metadata memory footprint of data reduction solutions [37, 66]. Additionally, endurance-aware SSD read-cache solutions have been proposed to reduce the impact of fragmentation on the read path of HDD-based storage [39]. All of these works target HDD-based backends and also evaluate the deduplication with caching on a single disk; thus, they do not analyze the interaction of RAID with data reduction, which is required in real SAN storage environments.

Some studies analyze the interaction of RAID and deduplication by mostly addressing reliability (not performance) [18, 19, 41, 53]. Other studies consider the performance and propose controlled data block replication across disks to provide more load-balanced I/O accesses for deduplication. Such works have mainly focused on I/O latency (not throughput) and evaluated their system using simulators [15]. In this paper, we use real measurements on enterprise-grade all-flash storage components and show the performance interaction of enterprise-grade data reduction with storage components (mainly RAID-controller). We further propose optimizations to the storage stack to boost the overall performance.

7.3 Data Reduction Hardware

Some works propose hardware acceleration platforms for data reduction on fast all-flash systems. These accelerations include GPU-based chunking for variable chunk boundary calculation [7], FPGA-based chunking [35], FPGA-based deduplication-only [6], FPGA-based compression-only [1, 17, 59], FPGA-based selected components for deduplication and compression [4, 5], ASIC-based hashing, duplicate verification, and metadata lookups [12]. Our analysis in this paper and the proposed optimizations are partly orthogonal to the data reduction implementation in software or hardware. In general, most studies that target hardware-based data reduction are mainly suitable for very high throughput environments (e.g., 10-100 GB/s). At common performance of SATA SSD arrays below 5 GB/s, recently introduced software libraries such as Intel ISA-L [23] usually provide a fast and cost-effective approach to implement the compute-intensive tasks of data reduction in software.

8 DISCUSSIONS

Generality of observations. In this paper, we explored major configuration parameters and two architectural optimization choices on a real storage system and showed how to build an all-flash storage stack with data reduction and almost optimal performance. We specifically focused on using RedHat VDO as a promising choice for enterprise-grade data reduction software. While our analysis of I/O overheads and performance results are partly dependent on the system implementation (including the choice of data reduction software), the conceptual effect of each design choice can be generalized to other all-flash storage systems.

Generality of proposed architecture. Our proposed architecture and the proposed optimizations have been implemented on an enterprise Linux software stack with the VDO module on the top of the device mapper layer; however, we believe the concepts and techniques we used are general enough to be applied to other data reduction modules and operating systems. In particular, although the implementation details are different across various platforms, our optimizations such as those for configuring the hardware RAID parameters, or accelerating VDO metadata (e.g., fingerprints and block map) accesses are conceptually applicable to any types of RAID controllers and enterprise-grade data reduction modules.

External I/O caching. One of the main goals in the design of an enterprise storage system is to provide a highly reliable software stack. In this regard, we use an I/O cache module outside VDO to provide a more reliable and easy-to-setup real solution, instead of trying to modify 150K lines of source code, which typically comes with numerous reliability issues in the software. Some older versions of VDO already have a read cache feature. However, such feature is removed in newer versions, and due to excessive DRAM usage, and also no-easy-access to some cache statistics, we adopted an external open-source I/O cache module.

Design for persistent memory. Our general goal in this paper was to accelerate the all-flash storage stack with integrated data reduction through modular software stack tuning and less intrusive kernel code modifications to facilitate its adoption in real environments that demand reliability. In this regard, our approach of offloading block maps and recovery journals to persistent memory involved minimal VDO source code modifications but more I/O redirection through external system modules. One may modify the source code more heavily and adopt some NVDIMM-specific optimization such as in-place metadata updates [65], to further improve our proposed system performance, but such optimizations have been left as future work.

Performance model. While this work manages to evaluate an industry-grade data reduction solution under a wide spectrum of workload patterns and system setups, developing a model to predict the system behavior beyond the examined system environment would also be of interest to designers. This model could provide a better understanding of the implications and how our findings are scaled. Meanwhile, developing a general model is challenging, as there are many hardware and software components and various parameters for each component that may contribute to the system performance with *non-linear* and *complex* interactions. A general performance model may consider three groups of hardware and three groups of software components. The hardware parts consist of a) disk subsystem: composed of SSDs and a backend controller, b) storage controller: composed of a RAID controller, a motherboard, DRAM modules, and processors, and c) network subsystem: composed of HBAs and SAN network. Software components can also be grouped into three parts: (a) data reduction and I/O cache modules, (b) storage stack including SSD device drivers and I/O schedulers, and (c) network stack including HBA device driver.

While some of these components are individually modeled in previous studies [9, 11, 26, 30, 31, 34, 48, 52, 55, 61, 62], due to huge per-component complexity, especially in data reduction and RAID layers, *no* comprehensive model covering the interaction of these components exists. Some full-system simulators such as MARSSx86 [48] can simulate the storage controller subsystem and interact with other network and disk simulators such as DiskSim [9], but simulators typically have many inaccuracies and also run slowly, offering no benefits over the real-system run in our case.

As a future work, one can work on a *coarse-grained* model considering the critical components of interest and parameters as the most practical solution in our case. Due to the large complexity of the mentioned parameters and their interactions, we suggest using *regression* analysis to estimate the relationship between the critical system parameters and the system performance. We can divide the parameters into unscalable and scalable parameters. For an arbitrary workload, one can consider independent regression models for every possible combination of unscalable parameters, while the scalable parameters are considered as the input vector. Accordingly, unscalable parameters and the number of their possible configurations (including RAID types, the write policy of RAID controller cache, and data reduction sync path) can result in too many possible combinations, which their discussion is beyond the scope of this paper.

9 CONCLUSION

Despite the complexity and importance of enterprise-grade data reduction on SSD arrays, no previous work has provided a detailed analysis of the impact of the data reduction module on architecting the underlying storage modules. In this paper, by employing the open-source, enterprise-grade data reduction software, RedHat VDO, we conducted an extensive set of experiments using various workload patterns on a real system. We revealed novel observations on the performance gap between the state-of-the-art and the optimal data reduction employment for all-flash storage systems. To fill this significant gap, we showed how to optimally configure the RAID subsystem, and also applied modifications to the storage stack architecture including the VDO source code and additional I/O caching layers to minimize the intrinsic enterprise-grade data reduction overheads. Our optimizations for write-intensive applications on all-flash arrays with integrated data reduction show up to 12.5× speedups over a non-optimized SSD array and up to 57× performance/cost improvement over an optimized SSD array with no data reduction.

ACKNOWLEDGMENTS

This work was funded by *High Performance Data Storage and Processing* (HPDS) Corporation, Tehran, Iran, and by the European *IO-SEA* project, which is supported by the *European High-Performance Computing Joint Undertaking* (JU) and the *German Ministry of Education and Research* (BMBF) under grant agreement number 955811.

REFERENCES

[1] Mohamed S. Abdelfattah, Andrei Hagiescu, and Deshanand P. Singh. 2014. Gzip on a chip: high performance lossless data compression on FPGAs using OpenCL. In *Proceedings of the International Workshop on OpenCL, IWOCL 2013 & 2014, May 13-14, 2013, Georgia Tech, Atlanta, GA, USA / Bristol, UK, May 12-13, 2014*. ACM, 4:1–4:9. https://doi.org/10.1145/2664666.2664670

[2] Saba Ahmadian, Onur Mutlu, and Hossein Asadi. 2018. ECI-Cache: A High-Endurance and Cost-Efficient I/O Caching Scheme for Virtualized Platforms. *Proc. ACM Meas. Anal. Comput. Syst.* 2, 1 (2018), 9:1–9:34. https://doi.org/10.1145/3179412

[3] Saba Ahmadian, Reza Salkhordeh, Onur Mutlu, and Hossein Asadi. 2021. ETICA: Efficient Two-Level I/O Caching Architecture for Virtualized Platforms. *IEEE Trans. Parallel Distributed Syst.* 32, 10 (2021), 2415–2433. https://doi.org/10.1109/TPDS.2021.3066308

[4] Mohammadamin Ajdari, Wonsik Lee, Pyeongsu Park, Joonsung Kim, and Jangwoo Kim. 2019. FIDR: A Scalable Storage System for Fine-Grain Inline Data Reduction with Efficient Memory Handling. In *Proceedings of the 52nd Annual IEEE/ACM International Symposium on Microarchitecture, MICRO 2019, Columbus, OH, USA, October 12-16, 2019*. ACM, 239–252. https://doi.org/10.1145/3352460.3358303

[5] Mohammadamin Ajdari, Pyeongsu Park, Joonsung Kim, Dongup Kwon, and Jangwoo Kim. 2019. CIDR: A Cost-Effective In-Line Data Reduction System for Terabit-Per-Second Scale SSD Arrays. In *25th IEEE International Symposium on High Performance Computer Architecture, HPCA 2019, Washington, DC, USA, February 16-20, 2019*. IEEE, 28–41. https://doi.org/10.1109/HPCA.2019.00025

[6] Mohammadamin Ajdari, Pyeongsu Park, Dongup Kwon, Joonsung Kim, and Jangwoo Kim. 2018. A Scalable HW-Based Inline Deduplication for SSD Arrays. *IEEE Comput. Archit. Lett.* 17, 1 (2018), 47–50. https://doi.org/10.1109/LCA.2017.2753258

[7] Pramod Bhatotia, Rodrigo Rodrigues, and Akshat Verma. 2012. Shredder: GPU-accelerated incremental storage and computation. In *Proceedings of the 10th USENIX conference on File and Storage Technologies, FAST 2012, San Jose, CA, USA, February 14-17, 2012*. USENIX Association, 14. https://www.usenix.org/conference/fast12/shredder-gpu-accelerated-incremental-storage-and-computation

[8] Broadcom Inc. 2022. *Knowledge Base*. Retrieved January 20, 2022 from https://www.broadcom.com/support/knowledgebase/1211161498420/performance-tuning-on-the-mr-sas-2108-lsi-sas-2208-sas-3108-base

[9] John S Bucy, Gregory R Ganger, et al. 2003. *The DiskSim simulation environment version 3.0 reference manual*. School of Computer Science, Carnegie Mellon University.

[10] Feng Chen, Tian Luo, and Xiaodong Zhang. 2011. CAFTL: A Content-Aware Flash Translation Layer Enhancing the Lifespan of Flash Memory based Solid State Drives. In *9th USENIX Conference on File and Storage Technologies, San Jose, CA, USA, February 15-17*. 77–90.

[11] Shenze Chen and Don Towsley. 1996. A performance evaluation of RAID architectures. *IEEE Transactions on computers* 45, 10 (1996), 1116–1130.

[12] Chris M. Evans. Jan 2017. HPE 3PAR Adaptive Data reduction: A competitive comparison of array-based data reduction. https://www.hpe.com/h20195/v2/getpdf.aspx/4AA6-6256ENW.pdf.

[13] Cornel Constantinescu, Joseph S. Glider, and David D. Chambliss. 2011. Mixing Deduplication and Compression on Active Data Sets. In *2011 Data Compression Conference (DCC 2011), 29-31 March 2011, Snowbird, UT, USA*. IEEE Computer Society, 393–402. https://doi.org/10.1109/DCC.2011.46

[14] Biplob K. Debnath, Sudipta Sengupta, and Jin Li. 2010. ChunkStash: Speeding Up Inline Storage Deduplication Using Flash Memory. In *USENIX Annual Technical Conference (ATC), Boston, MA, USA, June 23-25*. https://www.usenix.org/conference/usenix-atc-10/chunkstash-speeding-inline-storage-deduplication-using-flash-memory

[15] Yimo Du, Youtao Zhang, and Nong Xiao. 2014. R-Dedup: Content Aware Redundancy Management for SSD-Based RAID Systems. In *43rd International Conference on Parallel Processing (ICPP), Minneapolis, MN, USA, September 9-12*. 111–120. https://doi.org/10.1109/ICPP.2014.20

[16] Ahmed El-Shimi, Ran Kalach, Ankit Kumar, Adi Ottean, Jin Li, and Sudipta Sengupta. 2012. Primary Data Deduplication - Large Scale Study and System Design. In *2012 USENIX Annual Technical Conference, Boston, MA, USA, June 13-15, 2012*. USENIX Association, 285–296. https://www.usenix.org/conference/atc12/technical-sessions/presentation/el-shimi

[17] Jeremy Fowers, Joo-Young Kim, Doug Burger, and Scott Hauck. 2015. A Scalable High-Bandwidth Architecture for Lossless Compression on FPGAs. In *23rd IEEE Annual International Symposium on Field-Programmable Custom Computing Machines, FCCM 2015, Vancouver, BC, Canada, May 2-6, 2015*. IEEE Computer Society, 52–59. https://doi.org/10.1109/FCCM.2015.46

[18] Min Fu, Shujie Han, Patrick P. C. Lee, Dan Feng, Zuoning Chen, and Yu Xiao. 2018. A Simulation Analysis of Redundancy and Reliability in Primary Storage Deduplication. *IEEE Trans. Comput.* 67, 9 (2018), 1259–1272. https://doi.org/10.1109/TC.2018.2808496

[19] Min Fu, Patrick P. C. Lee, Dan Feng, Zuoning Chen, and Yu Xiao. 2016. A simulation analysis of reliability in primary storage deduplication. In *2016 IEEE International Symposium on Workload Characterization, IISWC 2016, Providence, RI, USA, September 25-27, 2016*. IEEE Computer Society, 199–208. https://doi.org/10.1109/IISWC.2016.7581280

[20] Fanglu Guo and Petros Efstathopoulos. 2011. Building a High-performance Deduplication System. In *USENIX Annual Technical Conference (ATC), Portland, OR, USA, June 15-17*. https://www.usenix.org/conference/usenixatc11/building-high-performance-deduplication-system

[21] Adam Horvath. 2021. *MurMurHash3, an ultra fast hash algorithm for C# / .NET.* Retrieved January 25, 2022 from https://blog.teamleadnet.com/2012/08/murmurhash3-ultra-fast-hash-algorithm.html

[22] Louis Imershein. 2018. Open Source Data Reduction for High Performance Flash Storage. In *Flash Memory Summit, 2018, Santa Clara, CA, USA*. https://www.flashmemorysummit.com/English/Collaterals/Proceedings/2018/20180808_SOFT-202-1_Imershein.pdf

[23] Intel. 2022. *Intel(R) Intelligent Storage Acceleration Library.* Retrieved January 20, 2022 from https://github.com/01org/isa-l

[24] Intel. 2022. *Open Cache Acceleration Software.* Retrieved January 20, 2022 from https://open-cas.github.io/

[25] Intel. 2022. *Open CAS Linux.* Retrieved January 20, 2022 from https://https://github.com/Open-CAS/open-cas-linux/

[26] Raj Jain and Shawn Routhier. 1986. Packet trains–measurements and a new model for computer network traffic. *IEEE journal on selected areas in Communications* 4, 6 (1986), 986–995.

[27] Tianyang Jiang, Guangyan Zhang, Zican Huang, Xiaosong Ma, Junyu Wei, Zhiyue Li, and Weimin Zheng. 2021. FusionRAID: Achieving Consistent Low Latency for Commodity SSD Arrays. In *19th USENIX Conference on File and Storage Technologies, FAST February 23-25, 2021*. USENIX Association, 355–370. https://www.usenix.org/conference/fast21/presentation/jiang

[28] Jürgen Kaiser, André Brinkmann, Tim Süß, and Dirk Meister. 2015. Deriving and comparing deduplication techniques using a model-based classification. In *Proceedings of the Tenth European Conference on Computer Systems, EuroSys 2015, Bordeaux, France, April 21-24, 2015*. ACM, 11:1–11:13. https://doi.org/10.1145/2741948.2741952

[29] Jürgen Kaiser, Tim Süß, Lars Nagel, and André Brinkmann. 2016. Sorted deduplication: How to process thousands of backup streams. In *32nd Symposium on Mass Storage Systems and Technologies, MSST, Santa Clara, CA, USA, May 2-6*. 1–14. https://doi.org/10.1109/MSST.2016.7897082

[30] Youngjae Kim, Brendan Tauras, Aayush Gupta, and Bhuvan Urgaonkar. 2009. Flashsim: A simulator for nand flash-based solid-state drives. In *2009 First International Conference on Advances in System Simulation*. IEEE, 125–131.

[31] Mostafa Kishani, Saba Ahmadian, and Hossein Asadi. 2020. A Modeling Framework for Reliability of Erasure Codes in SSD Arrays. *IEEE Trans. Computers* 69, 5 (2020), 649–665. https://doi.org/10.1109/TC.2019.2962691

[32] Andy Klein. 2022. *What SMART Hard Disk Errors Actually Tell Us.* Retrieved January 20, 2022 from https://www.backblaze.com/blog/what-smart-stats-indicate-hard-drive-failures

[33] Ricardo Koller and Raju Rangaswami. 2010. I/O Deduplication: Utilizing Content Similarity to Improve I/O Performance. In *8th USENIX Conference on File and Storage Technologies, San Jose, CA, USA, February 23-26, 2010*. USENIX, 211–224. http://www.usenix.org/events/fast10/tech/full_papers/koller.pdf

[34] Edward D Lazowska, John Zahorjan, G Scott Graham, and Kenneth C Sevcik. 1984. *Quantitative system performance: computer system analysis using queueing network models.* Prentice-Hall, Inc.

[35] Dongyang Li, Qing Yang, Qingbo Wang, Cyril Guyot, Ashwin Narasimha, Dejan Vucinic, and Zvonimir Bandic. 2015. A Parallel and Pipelined Architecture for Accelerating Fingerprint Computation in High Throughput Data Storages. In *23rd IEEE Annual International Symposium on Field-Programmable Custom Computing Machines, FCCM 2015, Vancouver, BC, Canada, May 2-6, 2015*. IEEE Computer Society, 203–206. https://doi.org/10.1109/FCCM.2015.43

[36] Huaicheng Li, Martin L. Putra, Ronald Shi, Xing Lin, Gregory R. Ganger, and Haryadi S. Gunawi. 2021. lODA: A Host/Device Co-Design for Strong Predictability Contract on Modern Flash Storage. In *SOSP ACM SIGOPS 28th Symposium on Operating Systems Principles, Virtual Event / Koblenz, Germany, October 26-29*. 263–279. https://doi.org/10.1145/3477132.3483573

[37] Wenji Li, Gregory Jean-Baptise, Juan Riveros, Giri Narasimhan, Tony Zhang, and Ming Zhao. 2016. CacheDedup: In-line deduplication for flash caching. In *14th {USENIX} Conference on File and Storage Technologies ({FAST} 16)*. 301–314.

[38] Mark Lillibridge, Kave Eshghi, Deepavali Bhagwat, Vinay Deolalikar, Greg Trezis, and Peter Camble. 2009. Sparse Indexing: Large Scale, Inline Deduplication Using Sampling and Locality. In *7th USENIX Conference on File and Storage Technologies, February 24-27, 2009, San Francisco, CA, USA. Proceedings*. 111–123. http://www.usenix.org/events/fast09/tech/full_papers/lillibridge/lillibridge.pdf

[39] Jian Liu, Yunpeng Chai, Xiao Qin, and Yao-Hong Liu. 2018. Endurable SSD-Based Read Cache for Improving the Performance of Selective Restore from Deduplication Systems. *Journal of Computer Science and Technology* 33, 1 (2018), 58–78. https://doi.org/10.1007/s11390-018-1808-5

[40] Sihang Liu, Korakit Seemakhupt, Gennady Pekhimenko, Aasheesh Kolli, and Samira Manabi Khan. 2019. Janus: optimizing memory and storage support for non-volatile memory systems. In *Proceedings of the 46th International Symposium on Computer Architecture, ISCA Phoenix, AZ, USA, June 22-26*. ACM, 143–156. https://doi.org/10.1145/3307650.3322206

[41] Tong Liu, Xubin He, Shakeel Alibhai, and Chentao Wu. 2018. Reference-Counter Aware Deduplication in Erasure-Coded Distributed Storage System. In *IEEE International Conference on Networking, Architecture and Storage (NAS), Chongqing, China, October 11-14*. 1–10. https://doi.org/10.1109/NAS.2018.8515697

[42] LSI Corporation. 2011. *LSI MegaRAID Advanced Software Evaluation Guide V3.0*. Retrieved January 20, 2022 from https://docs.broadcom.com/doc/12350183

[43] Lou Lydiksen. 2015. *Modeling workload IO size mixes with Oracle's Vdbench tool*. Retrieved January 20, 2022 from https://blog.purestorage.com/purely-technical/modeling-io-size-mixes-with-vdbench/

[44] Bo Mao, Hong Jiang, Suzhen Wu, and Lei Tian. 2014. POD: Performance Oriented I/O Deduplication for Primary Storage Systems in the Cloud. In *IEEE 28th International Parallel and Distributed Processing Symposium, Phoenix, AZ, USA, May 19-23*. 767–776. https://doi.org/10.1109/IPDPS.2014.84

[45] Dirk Meister and André Brinkmann. 2010. dedupv1: Improving deduplication throughput using solid state drives (SSD). In *IEEE 26th Symposium on Mass Storage Systems and Technologies, MSST 2012, Lake Tahoe, Nevada, USA, May 3-7*. 1–6. https://doi.org/10.1109/MSST.2010.5496992

[46] Fan Ni, Xingbo Wu, Weijun Li, and Song Jiang. 2018. ThinDedup: An I/O Deduplication Scheme that Minimizes Efficiency Loss due to Metadata Writes. In *2018 IEEE 37th International Performance Computing and Communications Conference (IPCCC)*. IEEE, 1–8.

[47] Oracle. 2019. *Architectural Overview of the Oracle ZFS Storage Appliance*. Retrieved January 20, 2022 from https://www.oracle.com/technetwork/server-storage/sun-unified-storage/documentation/o14-001-architecture-overview-zfsa-2099942.pdf

[48] Avadh Patel, Furat Afram, and Kanad Ghose. 2011. Marss-x86: A qemu-based micro-architectural and systems simulator for x86 multicore processors. In *1st International Qemu Users' Forum*. Citeseer, 29–30.

[49] David A. Patterson, Garth A. Gibson, and Randy H. Katz. 1988. A Case for Redundant Arrays of Inexpensive Disks (RAID). In *Proceedings of the 1988 ACM SIGMOD International Conference on Management of Data, Chicago, Illinois, USA, June 1-3, 1988*. 109–116. https://doi.org/10.1145/50202.50214

[50] RedHat. 2021. *dm-vdo*. Retrieved January 20, 2022 from https://github.com/dm-vdo/

[51] RedHat, Inc. 2021. *RedHat Enterprise Linux 8 Deduplicating and Compressing Storage: Using VDO to Optimize Storage Capacity in RHEL 8*. Retrieved January 20, 2022 from https://access.redhat.com/documentation/en-us/red_hat_enterprise_linux/8/pdf/deduplicating_and_compressing_storage/red_hat_enterprise_linux-8-deduplicating_and_compressing_storage-en-us.pdf

[52] Mendel Rosenblum, Stephen A Herrod, Emmett Witchel, and Anoop Gupta. 1995. Complete computer system simulation: The SimOS approach. *IEEE Parallel & Distributed Technology: Systems & Applications* 3, 4 (1995), 34–43.

[53] Eric Rozier, William H. Sanders, Pin Zhou, NagaPramod Mandagere, Sandeep Uttamchandani, and Mark L. Yakushev. 2011. Modeling the Fault Tolerance Consequences of Deduplication. In *30th IEEE Symposium on Reliable Distributed Systems (SRDS), Madrid, Spain, October 4-7*. 75–84. https://doi.org/10.1109/SRDS.2011.18

[54] Reza Salkhordeh, Mostafa Hadizadeh, and Hossein Asadi. 2019. An Efficient Hybrid I/O Caching Architecture Using Heterogeneous SSDs. *IEEE Trans. Parallel Distributed Syst.* 30, 6 (2019), 1238–1250. https://doi.org/10.1109/TPDS.2018.2883745

[55] Reza Salkhordeh, Onur Mutlu, and Hossein Asadi. 2019. An Analytical Model for Performance and Lifetime Estimation of Hybrid DRAM-NVM Main Memories. *IEEE Trans. Computers* 68, 8 (2019), 1114–1130. https://doi.org/10.1109/TC.2019.2906597

[56] Samsung. 2022. *Communicating With Your SSD: Understanding SMART Attributes*. Retrieved January 20, 2022 from https://www.samsung.com

[57] Sam Silverberg. 2022. *OpenDedup Overview*. Retrieved January 20, 2022 from https://opendedup.org/odd/overview/

[58] Kiran Srinivasan, Timothy Bisson, Garth R. Goodson, and Kaladhar Voruganti. 2012. iDedup: latency-aware, inline data deduplication for primary storage. In *Proceedings of the 10th USENIX conference on File and Storage Technologies, FAST 2012, San Jose, CA, USA, February 14-17, 2012*. USENIX Association, 24. https://www.usenix.org/conference/

fast12/idedup-latency-aware-inline-data-deduplication-primary-storage

[59] Bharat Sukhwani, Bülent Abali, Bernard Brezzo, and Sameh W. Asaad. 2011. High-Throughput, Lossless Data Compression on FPGAs. In *IEEE 19th Annual International Symposium on Field-Programmable Custom Computing Machines, FCCM 2011, Salt Lake City, Utah, USA, 1-3 May 2011*. IEEE Computer Society, 113–116. https://doi.org/10.1109/FCCM.2011.56

[60] Vasily Tarasov, Deepak Jain, Geoff Kuenning, Sonam Mandal, Karthikeyani Palanisami, Philip Shilane, Sagar Trehan, and Erez Zadok. 2014. Dmdedup: Device mapper target for data deduplication. In *2014 Ottawa Linux Symposium (OLS)*.

[61] Mojtaba Tarihi, Hossein Asadi, and Hamid Sarbazi-Azad. 2015. DiskAccel: Accelerating Disk-Based Experiments by Representative Sampling. In *Proceedings of the 2015 ACM SIGMETRICS International Conference on Measurement and Modeling of Computer Systems, Portland, OR, USA, June 15-19, 2015*. ACM, 297–308. https://doi.org/10.1145/2745844.2745856

[62] Mojtaba Tarihi, Soheil Azadvar, Arash Tavakkol, Hossein Asadi, and Hamid Sarbazi-Azad. 2022. Quick Generation of SSD Performance Models Using Machine Learning. *IEEE Transactions on Emerging Topics in Computing* (2022). https://doi.org/10.1109/TETC.2021.3116197

[63] The kernel development community. 2022. *BTRFS Main Page*. Retrieved January 20, 2022 from https://btrfs.wiki.kernel.org/index.php/Main_Page

[64] The kernel development community. 2022. *dm-linear*. Retrieved January 20, 2022 from https://www.kernel.org/doc/html/latest/admin-guide/device-mapper/linear.html

[65] Chundong Wang, Qingsong Wei, Jun Yang, Cheng Chen, Yechao Yang, and Mingdi Xue. 2017. NV-Dedup: High-performance inline deduplication for non-volatile memory. In *IEEETransactions on Computers*, Vol. 67. IEEE, 658–671.

[66] Qiuping Wang, Jinhong Li, Wen Xia, Erik Kruus, Biplob Debnath, and Patrick P. C. Lee. 2020. Austere Flash Caching with Deduplication and Compression. In *2020 USENIX Annual Technical Conference, USENIX ATC 2020, July 15-17, 2020*. USENIX Association, 713–726. https://www.usenix.org/conference/atc20/presentation/wang-qiuping

[67] Avani Wildani, Ethan L. Miller, and Ohad Rodeh. 2013. HANDS: A heuristically arranged non-backup in-line deduplication system. In *29th IEEE International Conference on Data Engineering, ICDE 2013, Brisbane, Australia, April 8-12, 2013*. IEEE Computer Society, 446–457. https://doi.org/10.1109/ICDE.2013.6544846

[68] Huijun Wu, Chen Wang, Yinjin Fu, Sherif Sakr, Liming Zhu, and Kai Lu. 2017. HPDedup: A Hybrid Prioritized Data Deduplication Mechanism for Primary Storage in the Cloud. *CoRR* abs/1702.08153 (2017). arXiv:1702.08153 http://arxiv.org/abs/1702.08153

[69] Suzhen Wu, Chunfeng Du, Weiwei Zhang, Bo Mao, and Hong Jiang. 2021. DedupHR: Exploiting Content Locality to Alleviate Read/Write Interference in Deduplication-based Flash Storage. In *IEEE Transactions on Computers, 2021*. IEEE Computer Society. https://doi.org/10.1109/TC.2021.3084116

[70] Wen Xia, Hong Jiang, Dan Feng, and Yu Hua. 2011. SiLo: A Similarity-Locality based Near-Exact Deduplication Scheme with Low RAM Overhead and High Throughput. In *2011 USENIX Annual Technical Conference, Portland, OR, USA, June 15-17, 2011*. USENIX Association. https://www.usenix.org/conference/usenixatc11/silo-similarity-locality-based-near-exact-deduplication-scheme-low-ram

[71] Wen Xia, Hong Jiang, Dan Feng, Lei Tian, Min Fu, and Zhongtao Wang. 2012. P-Dedupe: Exploiting Parallelism in Data Deduplication System. In *7th IEEE International Conference on Networking, Architecture, and Storage (NAS), Xiamen, China, June 28-30*. 338–347. https://doi.org/10.1109/NAS.2012.46

[72] Zhichao Yan, Hong Jiang, Song Jiang, Yujuan Tan, and Hao Luo. 2019. SES-Dedup: a case for low-cost ECC-based SSD deduplication. In *2019 35th Symposium on Mass Storage Systems and Technologies (MSST)*. 292–298. https://doi.org/10.1109/MSST.2019.00009

[73] Nannan Zhao, Hadeel Albahar, Subil Abraham, Keren Chen, Vasily Tarasov, Dimitrios Skourtis, Lukas Rupprecht, Ali Anwar, and Ali Raza Butt. 2020. DupHunter: Flexible High-Performance Deduplication for Docker Registries. In *USENIX Annual Technical Conference, USENIX ATC , July 15-17*. 769–783. https://www.usenix.org/conference/atc20/presentation/zhao

[74] Benjamin Zhu, Kai Li, and R. Hugo Patterson. 2008. Avoiding the Disk Bottleneck in the Data Domain Deduplication File System. In *6th USENIX Conference on File and Storage Technologies (FAST), San Jose, CA, USA, February 26-29*. 269–282. http://www.usenix.org/events/fast08/tech/zhu.html

[75] Xiangyu Zou, Jingsong Yuan, Philip Shilane, Wen Xia, Haijun Zhang, and Xuan Wang. 2021. The Dilemma between Deduplication and Locality: Can Both be Achieved?. In *19th USENIX Conference on File and Storage Technologies, FAST February 23-25*. USENIX Association, 171–185. https://www.usenix.org/conference/fast21/presentation/zou

[76] Pengfei Zuo, Yu Hua, Ming Zhao, Wen Zhou, and Yuncheng Guo. 2018. Improving the performance and endurance of encrypted non-volatile main memory through deduplicating writes. In *2018 51st Annual IEEE/ACM International Symposium on Microarchitecture (MICRO)*. IEEE, 442–454.

Received February 2022; revised March 2022; accepted April 2022