Elastic RAID: Implementing RAID over SSDs with Built-in Transparent Compression

Zheng Gu
ScaleFlux Inc.
zheng.gu@scaleflux.com

Jiangpeng Li
ScaleFlux Inc.
jiangpeng.li@scaleflux.com

Yong Peng
ScaleFlux Inc.
yong.peng@scaleflux.com

Yang Liu
ScaleFlux Inc.
yang.liu@scaleflux.com

Tong Zhang
ScaleFlux Inc. and Rensselaer Polytechnic Institute (RPI)
tzhang@ecse.rpi.edu

ABSTRACT

This paper studies how RAID (redundant array of independent disks) could take full advantage of modern SSDs (solid-state drives) with built-in transparent compression. In current practice, RAID users are forced to choose a specific RAID level (e.g., RAID 10 or RAID 5) with a fixed storage cost vs. speed performance trade-off. The commercial market is witnessing the emergence of a new family of SSDs that can internally perform hardware-based lossless compression on each 4KB LBA (logical block address) block, transparent to host OS and user applications. Beyond straightforwardly reducing the RAID storage cost, such modern SSDs make it possible to relieve RAID users from being locked into a fixed storage cost vs. speed performance trade-off. In particular, RAID systems could opportunistically leverage higher-than-expected runtime user data compressibility to enable dynamic RAID level conversion to improve the speed performance without compromising the effective storage capacity. This paper presents techniques to enable and optimize the practical implementation of such *elastic* RAID systems. We implemented a Linux software-based elastic RAID prototype that supports dynamic conversion between RAID 5 and RAID 10. Compared with a baseline software-based RAID 5, under sufficient runtime data compressibility that enables the conversion from RAID 5 to RAID 10 over 60% of user data, the elastic RAID could improve the 4KB random write IOPS (I/O per second) by 42% and 4KB random read IOPS in degraded mode by 46%, while maintaining the same effective storage capacity.

CCS CONCEPTS

• **Computer systems organization** → **Reliability**.

KEYWORDS

RAID, SSD, transparent compression

SYSTOR '23, June 5–7, 2023, Haifa, Israel

ACM ISBN 978-1-4503-9962-3/23/06…$15.00
https://doi.org/10.1145/3579370.3594773

ACM Reference Format:
Zheng Gu, Jiangpeng Li, Yong Peng, Yang Liu, and Tong Zhang. 2023. Elastic RAID: Implementing RAID over SSDs with Built-in Transparent Compression. In *The 16th ACM International Systems and Storage Conference (SYSTOR '23), June 5–7, 2023, Haifa, Israel.* ACM, New York, NY, USA, 11 pages. https://doi.org/10.1145/3579370.3594773

1 INTRODUCTION

This paper studies the implementation of RAID (redundant array of independent disks) [11, 34] over SSDs (solid-state drives) with built-in transparent data compression. As one of the best-known computing system design techniques, RAID plays an important role in building reliable computing infrastructure. In current practice, when deploying a RAID system, users must choose (and subsequently stick with) one specific RAID level after, often painfully, deliberating the trade-off between the storage cost and speed performance. For example, between RAID 10 and RAID 5, the two most popular RAID levels, RAID 10 achieves a higher I/O speed performance, in terms of IOPS (I/O requests per second) and average/tail latency, at the penalty of a higher data storage cost, while RAID 5 reduces the data storage cost by sacrificing the I/O speed performance. Such a storage cost vs. speed performance trade-off is inherent in the design of RAID, regardless of whether its implementation is software-based (e.g., Linux mdraid [29] and Btrfs RAID [38]) or hardware-based (e.g., RAID controller card [5]).

The commercial market currently witnesses the rise of SSDs with the built-in transparent data compression capability [20, 40]. Such modern SSDs internally carry out hardware-based compression on each 4KB LBA (logical block address) block, and could expose a logical storage space that is (much) larger than their internal physical NAND flash memory storage capacity. Evidently, one could deploy a RAID system (regardless of its RAID level) over such SSDs to reduce the storage cost without any changes to the RAID implementation and any degradation of the RAID speed performance. This paper shows that, beyond straightforwardly reducing the storage cost, SSDs with built-in transparent compression bring a unique opportunity to improve the RAID speed performance by elastically mixing different RAID levels (e.g., RAID 5 and RAID 10) in adaptation to the runtime user data compressibility variations. The basic idea can be described as follows: Suppose we deploy a RAID 5 over multiple SSDs with a total physical storage capacity of 32TB and format the RAID logical storage capacity as 64TB, i.e., we expect that the average data compressibility is about 2:1 and hence

aim at leveraging such SSDs to reduce the effective RAID storage cost by up to 2×. The storage system must adjust its monitoring and management accordingly to prevent out-of-space failure under worse-than-expected data compressibility. If the runtime data compressibility exceeds 2:1, the RAID system could *opportunistically* convert the protection of some user data from RAID 5 to RAID 10 in order to improve the RAID speed performance (especially in the degraded mode when one SSD is offline due to catastrophic failures), while still maintaining the total 64TB effective RAID data storage capacity. As the runtime data compressibility dynamically varies, the RAID system adaptively adjusts the mixture of RAID 5 and RAID 10. Such an *elastic RAID* design strategy opportunistically utilizes the runtime *residual* data compressibility to improve the speed performance without compromising the effective storage capacity. Elastic RAID can dynamically mix RAID 6 and triple replication if users demand double drive failure protection. For the purpose of simplicity, this paper focuses on the case of elastically mixing RAID 5 and RAID 10 in one RAID system, and the proposed design solutions could be readily extended to the case of elastically mixing RAID 6 and triple replication.

In spite of its simple concept, the practical implementation of elastic RAID is nontrivial. RAID 5 and RAID 10 have different data mappings and occupy different amounts of storage capacity, which makes it challenging to dynamically convert between RAID 5 and RAID 10 on the same array of SSDs at the minimal conversion-induced overhead in terms of data copy/move/delete operations. Moreover, we must retain the drive failure protection during the RAID level conversion. This paper presents a *bloated stripe allocation* method to facilitate the implementation of dynamic RAID level conversion at the minimal operational overhead. This paper further presents a strategy to schedule RAID level conversion both *proactively* in adaptation to workload characteristics variation and *reactively* in response to runtime data compressibility change. For the purpose of demonstration, we implemented a Linux software elastic RAID prototype in support of the mixture of RAID 5 and RAID 10. This prototype was developed by modifying/enhancing the existing Linux mdraid [29] to incorporate the proposed design techniques and meanwhile enhance the support of multi-threaded operations. We carried out experiments by deploying the software elastic RAID over commercial SSDs with built-in transparent compression from ScaleFlux Inc. [40]. We applied the widely used FIO (flexible I/O tester) tool [19] to generate heavy I/O workloads and collect the IOPS and tail latency results. When operating in the RAID 5 only mode, our elastic RAID implementation could noticeably outperform the state-of-the-art software RAID 5 product RAIDIX [37], and both RAIDIX and our elastic RAID achieve ~ 10× higher IOPS than the Linux mdraid. We further carried out experiments to evaluate the effect of elastic RAID 5 and RAID 10 mixture, and the results demonstrate its efficacy in improving the RAID speed performance without compromising the effective RAID storage capacity. For example, compared with the baseline that operates in the RAID 5 only mode, converting 20% and 60% user data from RAID 5 to RAID 10 could improve the 4KB random write IOPS by 10% and 42%, respectively. When the RAID system operates in the degraded mode (i.e., one SSD is offline), converting 20% and 60% RAID 10 user data from RAID 5 to RAID 10 could improve the 4KB random read IOPS by 12% and 46%, respectively. The experimental results show that a small increase in user data compression ratio could enable a significant increase in RAID 10 coverage. For example, the RAID 10 coverage could improve by over 40% if the user data compression ratio slightly increases from 1.2 to 1.4. The experimental results also show that the RAID level conversion can be carried out with very high throughput and its impact on the RAID system speed performance is very small (e.g., less than 5%) even under very heavy foreground user I/O workloads. This work demonstrates that emerging SSDs with built-in transparent compression make it practically feasible for a RAID system to opportunistically mix different RAID levels to improve the speed performance without compromising the effective data storage capacity.

2 BACKGROUND

2.1 Journaling in RAID

The basic design principle of RAID has been very well discussed in the open literature [11, 34, 41, 45]. Since our elastic RAID prototype heavily utilizes journaling to improve reliability and speed performance, this subsection briefly discusses the use of journaling in RAID systems. For RAID levels (e.g., RAID 5/6) that involve parity calculation, a partial-stripe data write incurs the read-modify-write operation to calculate the new parity. To hide the long latency of read-modify-write operations from end users, RAID systems could first log the incoming user data into a journal, and then carry out read-modify-write operations to update the parity in the background. Since RAID systems could utilize the journal to amortize the parity-update overhead, user data may stay inside the journal for a relatively long time. Hence, it is desirable to apply RAID 5/6 protection over the journal-resident user data as well. The append-only nature of journaling makes the implementation straightforward. Before being written into its destined stripe on the storage drive, the new parity should also be logged into the journal to guarantee atomic stripe update and hence obviate the potential write-hole problem.

For RAID 1/10 systems that do not involve parity calculation, they could also use journaling to reduce the write completion acknowledgment latency and ensure write atomicity. Upon receiving a write request, we first log the data into the RAID-protected journal and then acknowledge write completion to the host. In the background, user data are moved from the journal into their destined locations in the RAID system. In the presence of data write temporal locality, data journaling could help to reduce the overall system write amplification.

2.2 In-Storage Transparent Compression

Fig. 1(a) illustrates an SSD with built-in transparent compression: Its controller SoC (system on chip) performs (de)compression on each 4KB LBA data block along the I/O path and manages the placement of all the post-compression variable-length data blocks on the NAND flash memory. The in-storage per-4KB data compression is transparent to the host that accesses the SSD as a normal block data storage device through a standard I/O interface (e.g., NVMe or SATA). The per-4KB (de)compression latency of the hardware engine inside the SSD controller SoC can be well below 5μs, which is over 10× shorter than the TLC/QLC NAND flash memory read latency (~50μs and above) and write latency (~1ms and above).

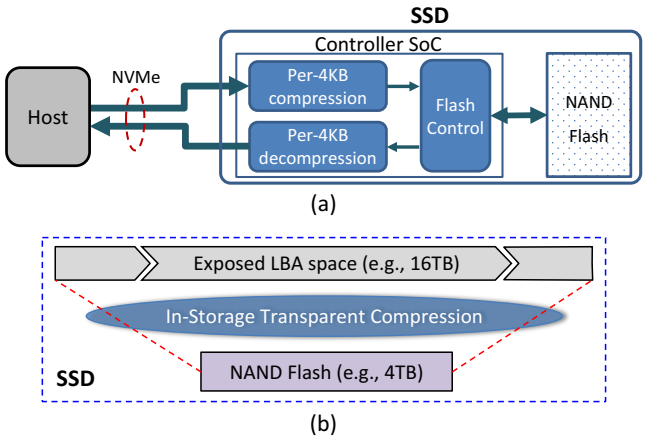

Figure 1: Illustration of (a) an SSD with built-in transparent compression, and (b) the expanded LBA space exposed by such SSDs.

Meanwhile, the hardware (de)compression engines could easily achieve a throughput higher than the aggregated bandwidth of back-end NAND flash memory chips (e.g., 4~8GB/s). Therefore, SSDs with built-in transparent compression can maintain the same IOPS and latency performance as traditional SSDs without built-in compression capability. In fact, by reducing the write stress on NAND flash memory through compression, such SSDs could have (much) lower GC (garbage collection) overhead, leading to (much) higher IOPS under write-intensive workloads. For example, under heavy 4KB random writes with 100% LBA span, traditional NVMe SSDs could achieve up to 200K~300K IOPS, while NVMe SSDs with built-in transparent compression (e.g., the one from ScaleFlux Inc. [40]) could achieve over 600K IOPS under 2:1 user data compression ratio.

To allow the host to materialize the benefit of in-storage transparent data compression, such modern SSDs could expose an expanded LBA logical storage space that is larger (e.g., by 2× or 4×) than its internal physical NAND flash memory storage capacity, as illustrated in Fig. 1(b). Given the runtime data compressibility variation, such SSDs with expanded LBA space may possibly run out of physical storage space before their exposed logical storage space has been used up by the host. Hence, to avoid running into the *out-of-space* error, the host must keep monitoring the SSD physical storage space usage and accordingly make its storage management aware of the runtime physical storage space usage, just like when using any thin-provisioned storage systems.

Finally, we note that data compression can be realized at different levels of the entire I/O stack (e.g., user applications, file system, block layer, and hardware storage devices). Whichever level handling data compression must meanwhile manage the storage of post-compression variable-length data blocks. For in-storage transparent compression, such a management task can be readily merged into the existing SSD FTL by enhancing SSD FTL to natively handle the storage of variable-length data blocks on NAND flash memory chips. However, when compression is performed at a higher level (e.g., file system), an extra layer of data mapping must be introduced to manage the storage of post-compression variable-length data blocks over normal storage devices, leading to non-negligible host CPU and memory usage overhead. Moreover, data compression at a higher level could consume significant host CPU resources for carrying out the data (de)compression computation. Modern all-flash array appliances (e.g., IBM Storwize [24] and Pure Storage FlashBlade [35]) widely support built-in hardware-based transparent compression and RAID data protection. However, such solutions result in higher $/GB costs than SSD in-storage compression. Therefore, for general-purpose RAID systems that can be deployed in commodity servers, it is highly desirable to employ SSDs with built-in transparent compression to reduce the storage cost without incurring any host CPU/memory overhead.

3 PROPOSED ELASTIC RAID

3.1 Rationale and Key Idea

When users deploy RAID over SSDs with built-in transparent compression, they should format their logical storage capacity based on the expected/estimated data compressibility in order to reduce the effective data storage cost. Let us consider RAID 5 over $n+1$ SSDs, and let C_{flash} denote the NAND flash memory capacity of each SSD (excluding its internal over-provisioned storage capacity reserved for GC). Throughout this paper, we define the compression ratio as the pre-compression data block size being divided by the post-compression data block size (i.e., a larger compression ratio corresponds to a higher data compressibility). Let $\alpha_{exp} \geq 1$ denote the expected average data compression ratio over the entire RAID 5 (including both user data and RAID 5 parity). Since all the user data and RAID 5 parity are evenly striped over the $n+1$ SSDs, all the SSDs will experience the same average data compression ratio. Hence, the logical storage capacity of the RAID 5 system should be formatted as $C_{RAID} = \alpha_{exp} \cdot n \cdot C_{flash}$, i.e., users expect to increase the effective RAID storage capacity by up to $\alpha_{exp}\times$ via deploying SSDs with built-in transparent compression. In practice, users tend to purposely underestimate the value of α_{exp} in order to better embrace unexpected data compressibility drop and hence reduce the probability of encountering out-of-space errors. Let α denote the runtime average compression ratio of all the data (both user data and parity) on a RAID 5 system. Once the runtime data compressibility is better than expected (i.e., $\alpha > \alpha_{exp}$), it will leave a certain amount of NAND flash memory storage space unused. We could utilize such *opportunistically available* physical data storage capacity to convert the protection of a portion (or even all) of user data from RAID 5 to RAID 10. This can contribute to improving the RAID I/O speed performance including IOPS and latency, while still maintaining the same RAID logical storage capacity of C_{RAID}.

Fig. 2 further illustrates the basic concept of such elastic RAID: To materialize the storage cost reduction enabled by deploying SSDs with built-in transparent compression, an elastic RAID system exposes a logical storage capacity of C_{RAID} to the host. When the runtime average data compression ratio α is less than α_{exp}, the elastic RAID system entirely operates as a classical RAID 5 to transform 100% of data compressibility into the storage cost reduction, as shown in Fig. 2. Being proportional to the average data compression ratio α, the effective data storage capacity that

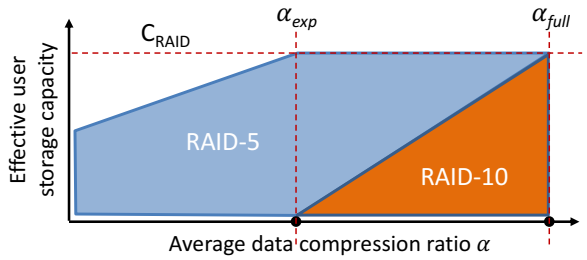

Figure 2: Illustration of elastic RAID enabled by in-storage transparent compression.

is truly consumable by the users is lower than the logical storage capacity C_{RAID} exposed by the elastic RAID. Hence, users must closely monitor the true physical storage capacity usage of SSDs in order to avoid out-of-space errors. Once the average runtime data compression ratio α reaches α_{exp}, the effective data storage capacity will reach C_{RAID} (i.e., all the logical storage space exposed by the RAID system could be truly consumed by the users). As α exceeds α_{exp}, the effective data storage capacity will remain as C_{RAID} and the residual data compressibility (i.e., $\alpha - \alpha_{exp}$) will be exploited to enable RAID 5 to RAID 10 conversion, as illustrated in Fig. 2, where α_{full} denotes the data compression ratio under which all the user data could be protected by RAID 10.

In summary, as the runtime average data compression ratio α dynamically varies between $[1, \alpha_{exp}]$, the effective data storage capacity of elastic RAID will proportionately change and all the user data are protected by RAID 5; as α dynamically varies between $[\alpha_{exp}, \alpha_{full}]$, elastic RAID will accordingly adjust the mixture of RAID 5 and RAID 10 while its effective data storage capacity remains as C_{RAID}; once α exceeds α_{full}, all the user data are protected by RAID 10 and the effective data storage capacity remains as C_{RAID}. Although elastic RAID is built upon a simple idea, its practical and efficient implementation poses unique challenges, which will be studied in the remainder of this section.

3.2 Realization of RAID Level Conversion

Compared with conventional RAID implementation, one unique challenge of elastic RAID is the realization and management of dynamic conversion between different RAID levels. Let $n+1$ denote the total number of SSDs in the RAID system, and $\mathcal{P}_i$ denote the LBA space exposed by the i-th SSD. Let $\mathcal{L}_{usr}$ denote the LBA space exposed by the RAID system to the user. The RAID system applies a mapping $f: \mathcal{L}_{usr} \rightarrow \{\mathcal{P}_1, \cdots, \mathcal{P}_{n+1}\}$ to manage the user data storage over the SSD array. With different amounts of data redundancy, different RAID levels must use different mapping functions f, leading to different user data placement over the $n+1$ SSDs. As a result, dynamic conversion between different RAID levels demands runtime varying the mapping function f (and hence the placement of user data and RAID parity). This could significantly complicate the data placement and management. Meanwhile, changing the mapping function will incur SSD operational overheads in terms of data copy/move/delete operations, leading to interference with foreground user I/O requests. Elastic RAID should reduce such operational overheads as much as possible to maximize its speed performance gain and reduce the impact on NAND flash memory endurance.

This work proposes a design technique to simplify the switching between different data mapping functions during the RAID level conversion. As pointed out above in Section 2.2, SSDs with built-in transparent compression could expose an expanded LBA space that is much larger than the physical NAND flash memory storage capacity. This enables the storage systems purposely *under-utilize* the SSD LBA space while still fully utilizing the physical NAND flash memory storage capacity. The key idea of the proposed design technique is to trade the SSD LBA space utilization efficiency for simpler data placement and management in support of dynamic RAID level conversion. Fig. 3 illustrates this design technique in the context of elastic RAID with RAID 5 and RAID 10. Given the total $n+1$ SSDs, each RAID 5 stripe contains n user data strips (denoted as $D_1, \cdots, D_n$) and one parity strip (denoted as P). We partition the entire LBA space of all the $n+1$ SSDs $\{\mathcal{P}_1, \cdots, \mathcal{P}_{n+1}\}$ into a large number of *segments*, where each segment is 2× larger than one RAID 5 stripe. As shown in Fig. 3, each segment is further partitioned into two equal-sized slots denoted as *slot-1* and *slot-2*. The two slots in each segment hold different content when the corresponding data stripe is protected by RAID 5 or RAID 10:

- In case of RAID 5, *slot-1* stores the entire RAID 5 stripe and *slot-2* is empty (hence all the LBA data blocks in *slot-2* are trimmed). The utilization of the expanded SSD LBA space is 50%.
- In case of RAID 10, *slot-1* and *slot-2* each stores one copy of stripe user data $D_1, \cdots, D_n$. In each slot, one strip is left unused and hence can be trimmed. The utilization of the expanded SSD LBA space is $n/(n+1)$.

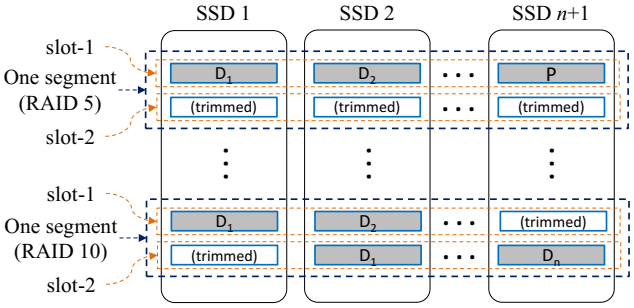

Figure 3: Illustration of the bloated stripe allocation to facilitate the conversion between RAID 5 and RAID 10.

By under-utilizing the SSD LBA space through the bloated stripe allocation, we can easily switch the data mapping function in support of dynamic RAID level conversion. Accordingly, the RAID system mapping function f can be decomposed into a large number of independent segment mapping functions $f^{(k)}: \mathcal{L}_{usr}^{(k)} \rightarrow \{\mathcal{P}_1^{(k)}, \cdots, \mathcal{P}_{n+1}^{(k)}\}$, where $\mathcal{L}_{usr}^{(k)}$ denotes the user data covered by one RAID stripe and $\mathcal{P}_i^{(k)}$'s correspond to the LBA space occupied by one segment. Within each segment, elastic RAID systems could conveniently realize dynamic conversion between RAID 5 and RAID 10:

- To convert one segment from RAID 5 to RAID 10, we first copy all the data strips D_i's from *slot-1* to *slot-2* in a skewed pattern so that two copies of the same data do not reside on the same SSD, and then trim the unused LBAs in *slot-1*.
- To convert one segment from RAID 10 to RAID 5, we first calculate the parity strip P based on the user data strips, write P to *slot-1*, and finally trim all the LBA data blocks in *slot-2*.

During the conversion between RAID 5 and RAID 10, we never in-place update any data blocks, which can ensure the atomicity of the conversion operation and hence keep data protected during the conversion. Moreover, since this design approach under-utilizes the SSD LBA space by up to 50%, given the target data compression ratio α_{exp}, each SSD should be formatted with an LBA space expansion factor of $2\alpha_{exp}$. Finally, we note that this proposed design technique only requires that the underlying SSDs can expose an LBA space larger than their internal physical storage space. This is essentially irrelevant to whether or not SSDs could carry out internal transparent compression. Therefore, this design technique is applicable to any RAID system as long as their SSDs are capable of exposing an expanded LBA space.

3.3 Management of RAID Level Conversion

Elastic RAID level conversion could be triggered *reactively* in response to data compressibility variation or *proactively* in adaptation to workload I/O data access characteristics variation. In either case, the elastic RAID controller should keep monitoring the SSD's internal physical NAND flash memory storage capacity usage. Since their internal FTL must keep track of the runtime NAND flash memory capacity usage, SSDs could make such information readily available to the host without incurring any extra SSD implementation overhead. To enable the host to obtain such runtime physical storage capacity usage information, one could add a new capacity query command into the block I/O interface protocol (e.g., NVMe) or SSD vendors could supply a software tool that can obtain such information from SSDs via proprietary commands over PCIe and expose a physical storage capacity query API to the host. In this work, we developed a software elastic RAID prototype based on ScaleFlux's SSDs with built-in transparent compression. ScaleFlux provides a user-space software tool through which the elastic RAID could query the runtime physical storage capacity usage of their SSDs.

Let us first consider the scenario of reactively triggering RAID level conversion in response to data compressibility variation. The above discussion, as illustrated in Fig. 2, suggests that all the user data should be protected by RAID 5 until the average data compression ratio α reaches α_{exp} under which the effective RAID storage capacity will reach C_{RAID}. This implicitly assumes that the users always utilize 100% of the available data storage capacity. Nevertheless, it is not uncommon that, in the real-world production environment, storage capacity utilization can be well below 100% (e.g., 80%∼90% and even lower) due to factors such as runtime storage usage fluctuation and storage capacity over-provisioning for an operational safety margin. Therefore, instead of triggering RAID level conversion only when $\alpha \in (\alpha_{exp}, \alpha_{full})$, we could opportunistically trigger RAID level conversion even under $\alpha < \alpha_{exp}$, in the case of users under-utilizing the available storage capacity. This can be realized by scheduling RAID level conversion directly based on the runtime physical storage capacity usage, other than the runtime data compression ratio. One simple scheduling approach is described as follows: The scheduler periodically samples the runtime physical storage capacity usage (denoted as C_{util}) in each SSD. Recall that C_{flash} denotes the total physical storage capacity of each SSD, and let C_u and C_l denote two pre-defined parameters (where $C_{flash} > C_u > C_l$). If C_{util} does not exhibit significant variation and remains below C_l in all the SSDs, then the scheduler will declare that the available storage capacity is being under-utilized and hence will trigger the conversion to promote some regions from RAID 5 to RAID 10. If $C_{util} > C_u$ in any SSD and elastic RAID currently contains RAID 10 regions, then the scheduler will trigger the conversion to demote some regions from RAID 10 to RAID 5 to restore sufficient operational safety margin for sudden storage capacity usage spikes.

To proactively trigger RAID level conversion in adaptation to data access locality variation, elastic RAID aims to ensure currently hot data are being covered by RAID 10. To simplify the implementation, we could partition the entire storage space into a number of equal-size regions (e.g., 10MB), and monitor the runtime data access intensity on a per-region basis. Within each region, all the RAID stripes are protected under the same RAID level (e.g., RAID 5 or RAID 10). Periodically we sort the per-region data access intensity, based on which we decide whether the RAID level setting of certain RAID 5 and RAID 10 regions should be flipped.

For both reactive and proactive RAID level conversion, the elastic RAID controller must carefully regulate their frequency. Too frequent RAID level conversion could incur unnecessary NAND flash memory read/write activities, leading to noticeable interference with normal user I/O requests. Moreover, since program/erase (P/E) cycling wears out the NAND flash memory cells, especially under highly scaled technology nodes, too frequent RAID level conversion could also severely accelerate the wear-out of NAND flash memory cells, leading to noticeable degradations of SSD endurance lifetime. Therefore, when scheduling RAID level conversion, elastic RAID should cohesively consider the wear-out of SSDs. As SSDs undergo more P/E cycles, elastic RAID should accordingly more and more throttle the frequency of RAID level conversion and even suspend the conversion at the end of the SSD lifetime. Elastic RAID controllers should also appropriately configure the throughput of ongoing RAID level conversion operations: On one hand, the conversion throughput should not be too high in order to avoid significantly interfering with normal user I/O requests. On the other hand, in the presence of sudden data compressibility drop, the RAID 10 to RAID 5 conversion throughput should be sufficiently high to avoid out-of-space errors.

Finally, we note that, in the presence of intra-SSD transparent compression, RAID 5 is not necessarily always more cost-effective than RAID 10. This is because user data and their RAID parity could have significantly different compressibility. For the purpose of demonstration, using two compression benchmark corpus files (i.e., file *kennedy* in the Canterbury corpus [8] and file *samba* in the Silesia corpus [42]) as representative user data, we measured the

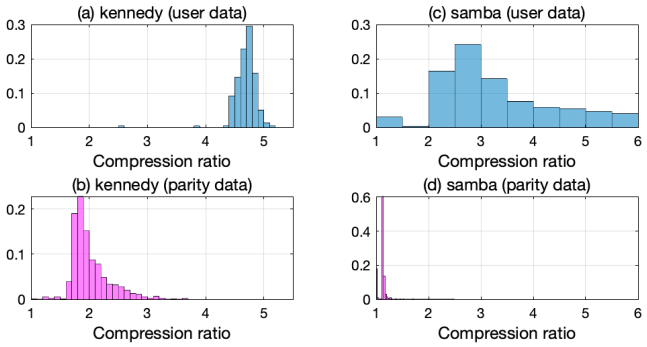

Figure 4: Histograms of per-4KB compression ratio of user data and RAID 5 parity when using two compression benchmark files in the Canterbury corpus [8] and Silesia corpus [42].

per-4KB compression ratio of both user data and RAID 5 parity (assuming 3+1 RAID 5). Fig. 4 shows the histogram of measured per-4KB compression ratio (using the GZIP compression library), which clearly reveals that RAID 5 parity has much worse compressibility than user data. For *samba*-based experiments, the RAID 5 parity data are even almost completely incompressible. This phenomenon can be explained as follows: Lossless data compression is mainly realized by deduplicating repeated byte strings in the data stream. Different user data strips within the same stripe most likely have repeated byte strings at different locations inside 4KB blocks. As a result, being obtained by bit-wise XOR over multiple user data strips, RAID parity tends to have much less amount of repeated byte strings, leading to much worse compressibility as shown in Fig. 4. Therefore, it is possible that, after converting a region from RAID 10 to RAID 5, the physical storage capacity usage increases other than decreases. In this case, we should revert the conversion and keep the region in the RAID 10 mode. Moreover, if SSDs support the host query the compression ratio on the per-LBA basis, we could scan each RAID 5 region to obtain the compression ratio of user data strips and RAID parity strips. Accordingly, we could calculate whether converting one stripe to a RAID 10 region could actually reduce the physical storage capacity usage and then decide whether we should proactively convert this RAID 5 region into a RAID 10 region.

4 PROTOTYPING AND EVALUATION

The proposed elastic RAID design strategy could be integrated into a software RAID solution or a dedicated hardware RAID card. For the purpose of demonstration, by modifying the existing Linux mdraid [29], we implemented a Linux software elastic RAID (SW eRAID) prototype that supports the dynamic mixture of RAID 5 and RAID 10. Following the discussion above in Section 2.1, it implements a write journal to mitigate the well-known write hole problem [34] of RAID 5 and meanwhile reduce the write request latency experienced by end users. The write journal spans over all the SSDs and is protected by RAID 5 as well. Because the write journal is append-only, its use of RAID 5 is not subject to the write-hole problem. Upon receiving a write request, SW eRAID flushes the data into the write journal and then immediately acknowledges the write completion to the user. Data are migrated from the write journal into their destined stripes asynchronously in the background, which is realized in batches with multiple threads in order to prevent background data migration from becoming the overall RAID speed performance bottleneck.

We carried out experiments on a server with a 26-core 2.5GHz Intel Xeon CPU, 565GB DRAM, and 4 commercial NVMe SSDs with built-in transparent compression from ScaleFlux Inc. [40]. The NVMe SSDs carry out hardware-based zlib [52] compression on each 4KB LBA data block directly along the I/O path. It can achieve a compression ratio similar to that of the zlib level-6. We applied the widely used FIO (flexible I/O tester) tool [19] to generate heavy foreground random data access I/O workloads and collect the IOPS and tail latency results. FIO can generate compressible data according to the target compression ratio specified by the users. The system OS is CentOS Linux release 7.6.1810, and the FIO version is 3.13. In all the experiments, RAID strip size is set as 4KB, and the RAID system spans over all the 4 SSDs.

4.1 Baseline RAID 5 Performance

We first carried out experiments to evaluate the baseline speed performance under random write workloads when the system operates in the RAID 5 only mode (i.e., all the data are protected by 3+1 RAID 5 over the 4 SSDs). For the purpose of comparison, we carried out the same experiments over the existing Linux mdraid and the state-of-the-art software RAID product RAIDIX [37]. We focused on 4KB random write workloads over the entire 100% LBA span to trigger the worst-case scenarios for RAID 5. To ensure sufficient I/O workload stress over all three RAID 5 systems, we configured FIO to run with 16 jobs at the I/O queue depth of 128. Table 1 lists the IOPS and tail latency results of the three different RAID 5 systems.

Table 1: RAID 5 under 4KB random writes.

	IOPS	Tail latency (ms)	
		99%	99.9%
Linux mdraid	59,003	40.1	43.8
RAIDIX	474,389	40.1	246.4
SW eRAID	677,829	9.6	14.9

The results show that RAIDIX and our SW eRAID can achieve one order of magnitude higher 4KB random write IOPS than Linux mdraid. This is mainly because Linux mdraid uses a single management thread to control the RAID 5 state machine. Although such single-thread management implementation works well on HDD-based RAID, it could easily become the speed performance bottleneck of SSD-based RAID. Compared with RAIDIX, our SW eRAID achieves 1.4× higher 4KB random write IOPS at 4.2× shorter 99% tail latency. Because we do not have access to the source code of RAIDIX, we conjecture that this is mainly due to the different implementation efficiency of background data movement from the write journal to destined stripes. The results demonstrate that our SW eRAID matches the state of the art of SW RAID implementations, hence it can be used to reliably evaluate the effectiveness of dynamically mixing RAID 5 and RAID 10.

4.2 RAID 5 vs. RAID 10 Performance

The RAID 5 vs. RAID 10 speed performance difference is most noticeable when serving (i) random data write requests or (ii) random read requests under degraded mode when one SSD is offline (e.g., one SSD suffers a catastrophic failure). This is due to the large RAID 5 vs. RAID 10 difference in terms of the read amplification under these two scenarios. Table 2 summarizes the write/read amplification of the SW eRAID when serving random writes under the normal mode and random reads under the degraded mode.

Table 2: Write/read amplification under random data access.

	Random write (normal mode)		Random read (degraded mode)
	WA	RA	RA
RAID 5	$2+(n+1)/n$	2	$2n/(n+1)$
RAID 10	$2+(n+1)/n$	0	1

WA: write amplification; RA: read amplification.

Table 3 lists the SW eRAID IOPS and tail latency under 4KB random write workloads. Again, to ensure sufficient I/O workload stress on SSDs, we configured FIO to run with 16 jobs at the I/O queue depth of 128. The results show that RAID 10 achieves 1.67× higher 4KB random write IOPS than RAID 5, while maintaining a similar tail latency. This is due to the significantly reduced read amplification of RAID 10 compared with RAID 5 under random write workloads.

Table 3: SW eRAID under 4KB random write.

RAID Mode	IOPS	Tail latency (ms)	
		99%	99.9%
RAID 5	677,829	9.6	14.9
RAID 10	1,138,546	10.0	16.5

If one SSD becomes offline, the RAID system will operate in the *degraded* mode and serve read requests only. Given the total 4 SSDs in the system, as shown above in Table 2, RAID 5 suffers from a read amplification of 1.6 under the degraded mode, while RAID 10 does not experience any read amplification. Table 4 lists the measured 4KB random read IOPS and tail latency when RAID 5 and RAID 10 operate in the degraded mode. To further stress the SSDs under read workloads, we increased the FIO job number from 16 to 32 while maintaining the same queue depth of 128. The results show that, compared with RAID 5, RAID 10 could achieve 1.75× higher read IOPS at 3.3× shorter 99% tail latency. The results demonstrate the speed performance advantage of RAID 10 over RAID 5, which motivates our proposed elastic RAID design strategy.

Table 4: 4KB random read in degraded mode.

RAID Mode	IOPS	Tail latency (ms)	
		99%	99.9%
RAID 5	1,434,445	14.5	22.9
RAID 10	2,505,795	4.4	4.8

4.3 SW eRAID with mixed RAID 5/10

Given sufficient runtime user data compressibility, SW eRAID could transparently convert data protection from RAID 5 to RAID 10 without sacrificing the effective RAID storage capacity. As discussed in Section 3.3, we partition the entire RAID storage space into equal-sized segments (10MB per segment in this study) and use the same RAID level for each segment. RAID 10 outperforms RAID 5 when serving random writes in normal mode and random reads in degraded mode. Fig. 5 shows the 4KB random write IOPS and tail latency when different percentages of data are protected by RAID 10. As user data compressibility improves, eRAID will convert the protection of more data from RAID 5 to RAID 10, which naturally leads to a higher speed performance shown in Fig. 5.

Fig. 6 shows the 4KB random read IOPS and tail latency under the degraded mode when different percentages of data are protected by RAID 10. As one could intuitively justify, the read speed performance will improve when user data compressibility increases, and hence more data are protected under RAID 10. As shown in Fig. 6, the 99% read tail latency largely drops even as the IOPS increases. This is because a higher percentage of RAID 10 leads to a lower read amplification under the degraded mode and hence a smaller read I/O queue depth, which directly contributes to the shorter read latency.

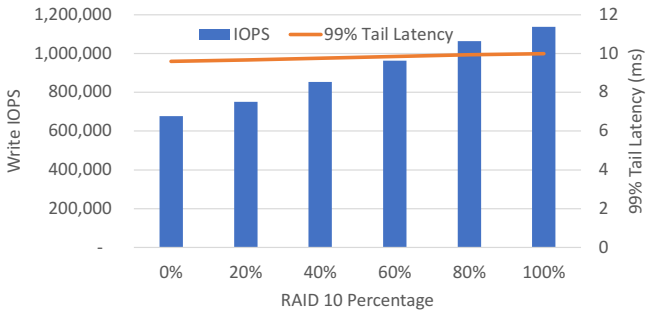

Figure 5: SW eRAID 4KB random write IOPS and 99% tail latency under different percentages of data being protected by RAID 10.

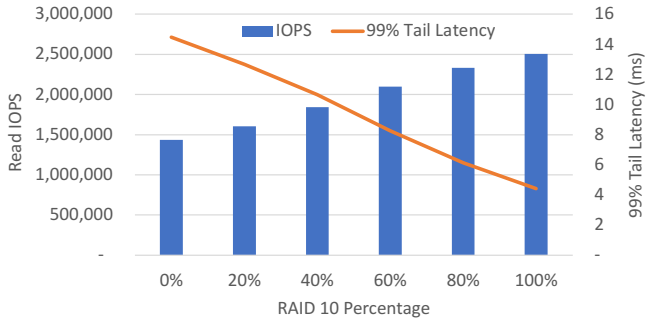

Figure 6: SW eRAID 4KB random read IOPS and 99% tail latency in the degraded mode under different percentages of data being protected by RAID 10.

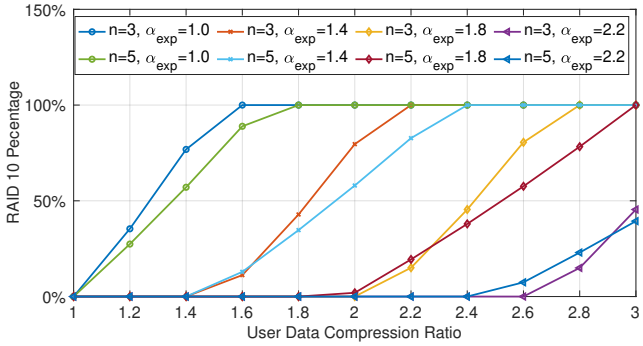

Figure 7: Percentage of RAID 10 under different storage capacity expansion factor α_{exp} when the storage capacity utilization factor β_{util} is 100%.

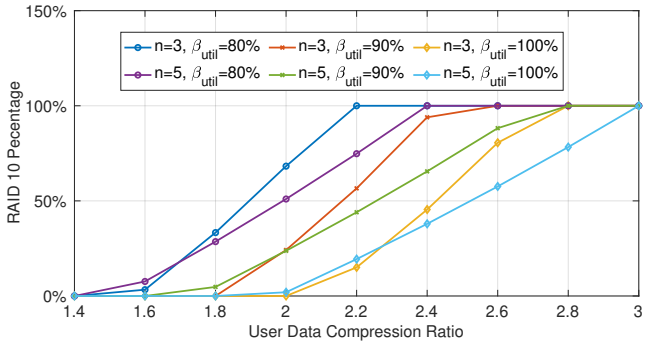

Figure 8: Percentage of RAID 10 under different storage capacity utilization factor β_{util} when the capacity expansion factor α_{exp} is 1.8.

The above results show that the performance advantage of SW eRAID heavily depends on the percentage (denoted as P_{conv}) of data being converted from RAID 5 to RAID 10. In addition to the user data compressibility, P_{conv} also depends on the storage capacity expansion factor $\alpha_{exp} \geq 1$ and storage capacity utilization factor $\beta_{util} \leq 1$. Given the total $n+1$ SSDs and per-SSD physical storage capacity of C_{flash}, eRAID exposes a total logical storage capacity of $\alpha_{exp} \cdot n \cdot C_{flash}$ to the user. As we increase α_{exp} to more aggressively leverage data compressibility to increase the effective RAID storage capacity, less amount of residual data compressibility will be left for RAID 5 to RAID 10 conversion. Meanwhile, as discussed in Section 3.3, users may not always utilize 100% of the storage capacity, and eRAID could accordingly schedule the RAID level conversion in response to the runtime storage capacity utilization factor β_{util}. Fig. 7 shows the percentage of RAID 10 under different storage capacity expansion factors α_{exp}. We considered the cases of $n = 3$ or $n = 5$ (i.e., the RAID system contains 4 or 6 SSDs). The storage capacity utilization factor β_{util} is set as 100% (i.e., users utilize the entire storage capacity exposed by the RAID system). As discussed in Section 3.3, RAID 5 parity tends to have a much worse compression ratio than user data. Let α_{usr} and α_{pty} denote the compression ratio of user data and RAID 5 parity, according to the results presented in Fig. 4, we set $(\alpha_{usr} - 1) = 4(\alpha_{pty} - 1)$ so that the RAID 5 parity is incompressible (i.e., $\alpha_{pty} = 1$) when user data is incompressible (i.e., $\alpha_{usr} = 1$), and RAID 5 parity compression ratio is 1.5 when user data compression ratio is 3. As shown in Fig. 7, when the storage capacity expansion factor α_{exp} increases, a larger user data compression ratio is required to enable RAID 5 to RAID 10 conversion. Under a larger storage capacity expansion factor α_{exp}, the RAID 10 percentage will increase more slowly with the user data compression ratio α_{usr}. For example, when $\alpha_{exp} = 1.0$ (i.e., users do not leverage data compression to reduce storage cost at all), the user data compression ratio only needs to increase from 1 to 1.6 in order to enable the 100% RAID 5 to RAID 10 conversion; when $\alpha_{exp} = 1.8$ (i.e., users expect to reduce the storage cost by up to 1.8× via data compression), the user data compression ratio must jump from 1.8 to 2.8 in order to enable 100% RAID 5 to RAID 10 conversion. The results also show the noticeable impact of the total number of SSDs on the relationship between user data compressibility and RAID 10 coverage percentage, i.e., the more SSDs are deployed in the RAID system, the slower the RAID 10 coverage percentage will grow with the user data compression ratio. This result stems from the fact that as the RAID system contains more SSDs, the storage efficiency gap between RAID 5 and RAID 10 will accordingly increase. Hence a higher data compression ratio is required to fill the gap.

Fig. 8 shows the percentage of RAID 10 under different storage utilization factors β_{util} when the storage capacity expansion factor α_{exp} is set to 1.8. The results show that under-utilized storage capacity could be effectively leveraged to improve the RAID 10 coverage percentage. For example, under the user data compression ratio of 2, the RAID 10 coverage percentage is almost 0% when the storage capacity utilization factor β_{util} is 100%, and it will improve to 25% and 68% when β_{util} reduces to 90% and 80%, respectively. Since it is not uncommon for a real production environment to have a storage capacity utilization factor of 80%~90% or even lower, the results suggest that the adaptive RAID level conversion scheduling could very noticeably improve the RAID 10 coverage percentage.

The above results suggest that a small increase in the user data compression ratio could enable a noticeable increase in RAID 10 coverage percentage. Meanwhile, real-world I/O workloads typically exhibit considerable degrees of access locality. As discussed above in Section 3.3, eRAID could schedule RAID level conversion in adaptation to the runtime data access locality. Fig. 9 shows the 4KB random read IOPS in the degraded mode under random workload (i.e., zero data access locality) and 80/20 skewed workload (i.e., 80% of reads hit 20% of data). The eRAID system storage capacity expansion factor α_{exp} is set to 1.4, and the RAID storage capacity utilization factor β_{util} is set to 100%. The RAID 10 coverage percentage increases from 0% to 100% when the user data compression ratio increases from 1.4 to 2.2. As shown in Fig. 9, under the 80/20 skewed workload, read IOPS could improve by 33% even when the user data compression ratio only increases from 1.4 to 1.6.

4.4 RAID Level Conversion

We further evaluated the RAID level conversion and its impact on RAID system I/O speed performance. Our SW eRAID prototype uses multiple background threads to carry out RAID level

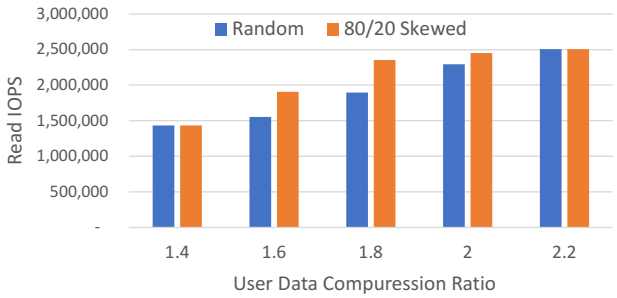

Figure 9: SW eRAID 4KB random read IOPS in the degraded mode under random and 80/20 skewed workloads.

conversion between RAID 5 and RAID 10. The RAID level conversion throughput improves as the number of background threads increases. Table 5 shows the RAID level conversion throughput in the absence of foreground user I/O requests. The user data compression ratio is 2:1. As discussed in Section 3.2, to convert one stripe from RAID 5 to RAID 10, we only need to duplicate the data once within the stripe segment; to convert one stripe from RAID 10 to RAID 5, we fetch the data stripe and calculate and write the RAID 5 parity. Hence, under our experimental RAID system with 4 SSDs, RAID 10 to RAID 5 conversion tends to have a lower throughput than RAID 5 to RAID 10 conversion, as shown in Table 5. The results suggest that the RAID level conversion can easily achieve multi-GB/s throughput, which demonstrates the efficiency of the proposed bloated stripe allocation strategy.

Table 5: RAID level conversion throughput.

Conversion mode	# of conversion threads	Throughput (GB/s)
RAID 5 to RAID 10	8	3.1
	24	6.4
RAID 10 to RAID 5	8	2.6
	24	5.1

Next, we evaluated the impact of background RAID level conversion on the speed performance of SW eRAID serving foreground user read/write requests. Evidently, we could control the performance impact by throttling the background RAID level conversion throughput. In the case of a sudden and significant drop in the average user data compression ratio within a short period, eRAID may have to accordingly adjust the RAID 10 to RAID 5 conversion throughput in order to avoid out-of-space errors. Meanwhile, the impact of background RAID level conversion also depends on the intensity of the foreground user read/write requests, i.e., the heavier the foreground I/O workloads are, the more noticeably they will be impacted by the background RAID level conversion. To examine the worst-case scenario, we use heavy 4KB random read or write workloads with 32 FIO jobs to emulate very heavy foreground user I/O workloads.

Fig. 10 and Fig. 11 show the impact of background RAID level conversion under three different throttled conversion throughputs, including 100MB/s, 200MB/s, and 400MB/s. The results show that, even under the very heavy 32-job FIO foreground I/O workloads, the impact of background RAID level conversion can be small (e.g., no more than 7% of IOPS drop with 200MB/s background RAID level conversion). The results further show that, compared with RAID 5 to RAID 10 conversion, RAID 10 to RAID 5 conversion causes a similar impact on random read speed performance but noticeably less impact on random write speed performance. This is because, compared with RAID 5 to RAID 10 conversion, RAID 10 to RAID 5 conversion generates much less amount of write traffic to SSDs.

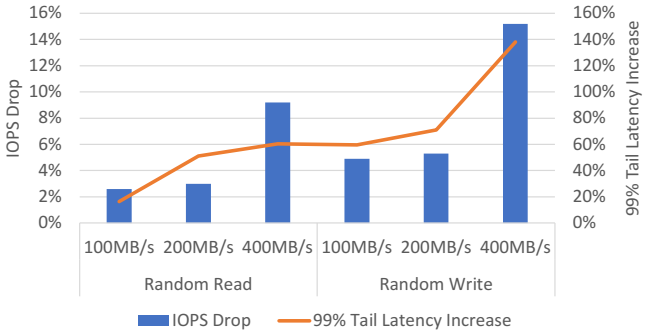

Figure 10: Impact of background RAID 5 to RAID 10 conversion on heavy random read or write workloads with 32 FIO jobs.

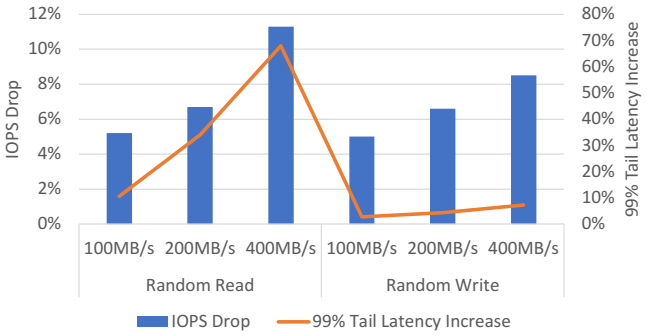

Figure 11: Impact of background RAID 10 to RAID 5 conversion on heavy random read or write workloads with 32 FIO jobs.

4.5 Impact of Inaccurate RAID Level Configuration

As discussed above in Section 3.3, we partition all the data into a number of regions and configure RAID level (e.g., RAID 5 vs. RAID 10) on the per-region basis according to the data access intensity history. Dynamic workload I/O characteristics variation could inevitably cause RAID level misconfiguration over some regions. We carried out further experiments to evaluate the impact of possible RAID level misconfiguration. We ran two FIO processes

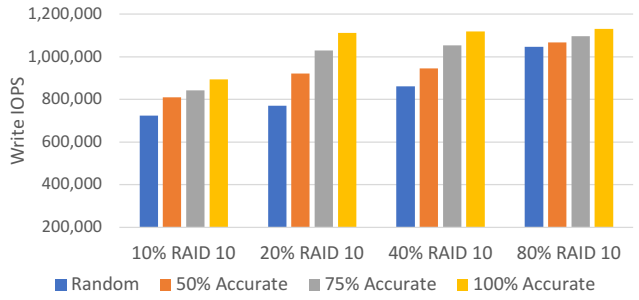

Figure 12: Impact of classification accuracy on 4KB random write IOPS of 80/20 skewed workloads.

to emulate 80/20 skewed 4KB random write workloads. Fig. 12 shows the overall random write IOPS under different RAID level configuration accuracy, e.g., 75% *accuracy* means 75% of write-hot data under the 80/20 skewed 4KB random write workloads are accurately classified as hot by the RAID system. For the purpose of comparison, we also considered a *Random* case where data are just randomly classified as write-hot. We studied four different data compressibility under which 10%, 20%, 40%, and 80% of data are protected by RAID 10. As the data classification accuracy improves, a higher percentage of write-hot data will be covered by RAID 10, leading to a higher write IOPS as shown in Fig. 12. Moreover, as data compressibility improves to enable a higher RAID 10 coverage, classification accuracy tends to have less impact on the overall IOPS. For example, when 20% of data can be protected by RAID 10, write IOPS could increase by 20.6% if the classification accuracy improves from 50% to 100%. In comparison, when 80% of data can be protected by RAID 10, increasing the classification accuracy from 50% to 100% improves write IOPS by only 6%.

5 RELATED WORK

RAID. As the most widely deployed data protection solution, RAID has been very well studied in the open literature. Prior work on RAID mainly focused on more accurately modeling and analyzing the RAID system reliability [2, 17, 18, 28, 30, 44], reducing the drive rebuild time and accelerating the data recovery process [21, 32, 47, 48, 51], and better embracing the device characteristics of SSDs [3, 9, 25, 31]. Prior research [33, 49] also studied the implementation of *tiered* RAID systems consisting of a hot-data RAID 10 tier and warm/cold-data RAID 5/6 tier. The well-known AutoRAID [49] has been commercialized by HP from the late 1990s to the late 2000s. Operating on traditional HDDs or SSDs, tiered RAID systems must employ complicated data management/migration strategies to realize dynamic tiering and embrace different data mapping functions of different RAID levels, and they are still fundamentally subject to the speed performance vs. storage cost trade-off. These limitations are at least part of the reason why HP terminated the AutoRAID product line over a decade ago. To the best of our knowledge, no prior work has ever studied the feasibility of opportunistically leveraging runtime data compressibility to enable a dynamic mixture of different RAID levels without sacrificing the effective data storage cost.

Transparent Data Reduction. Prior research has well studied the implementation of storage data reduction (e.g., compression and deduplication) with complete transparency to user applications. Transparent data reduction can be realized at the filesystem level [4, 6, 38, 43], block level [1, 27, 46], and even inside storage hardware [10, 12, 50, 53]. Modern all-flash array products (e.g., Dell EMC PowerMAX [16], HPE Nimble Storage [22], and Pure Storage FlashBlade [35]) always come with built-in hardware-based transparent compression capability. Cloud vendors have started to integrate hardware-based compression capability into their storage infrastructure, e.g., Microsoft Corsia [14] and emerging DPU (data processing unit) [7]. Motivated by the emergence of storage hardware with built-in transparent compression, researchers have recently studied its implications for the design of hash-based key-value store [13] and B-tree [36].

SSD Sparse Addressing. Prior work studied how one could innovate the data management systems by making SSD expose a sparse logical address space that is (much) larger than its internal physical storage capacity. FlashTier [39] utilizes the SSD sparse addressing to largely simplify the SSD cache management. FlashMap [23] integrates virtual address translation and SSD FTL sparse address translation to efficiently support memory-mapped SSD files. Das et al. [15] presented a solution that leverages SSD sparse addressing to facilitate application-level data compression. DFS filesystem [26] takes advantage of SSD sparse addressing to significantly simplify its data management.

6 CONCLUSIONS

This paper presents an elastic RAID design strategy to take full advantage of modern SSDs with built-in transparent compression capability. The key idea is to leverage the opportunistically available residual data compressibility to enable dynamic RAID 5 to RAID 10 conversion, which can improve the RAID system speed performance without sacrificing its effective RAID storage capacity. This paper presents design techniques for efficiently implementing elastic RAID at the minimal RAID level conversion overhead. We implemented a Linux software-based elastic RAID prototype in support of dynamic conversion between RAID 5 and RAID 10, and experimental results demonstrate the effectiveness of the proposed elastic RAID design solution.

ACKNOWLEDGMENTS

We thank our shepherd Aviad Zuck and the anonymous reviewers for their insightful and constructive comments. The co-author T. Zhang was supported in part by the National Science Foundation (NSF) under grants CNS-2006617 and CCF-2210754.

REFERENCES

[1] 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 *IEEE International Symposium on High Performance Computer Architecture (HPCA)*. IEEE, 28–41.

[2] Sung Hoon Baek, Bong Wan Kim, Eui Joung Joung, and Chong Won Park. 2001. Reliability and performance of hierarchical RAID with multiple controllers. In *ACM symposium on Principles of Distributed Computing*. 246–254.

[3] Mahesh Balakrishnan, Asim Kadav, Vijayan Prabhakaran, and Dahlia Malkhi. 2010. Differential raid: Rethinking raid for SSD reliability. *ACM Transactions on Storage (TOS)* 6, 2 (2010), 1–22.

[4] Jeff Bonwick, Matt Ahrens, Val Henson, Mark Maybee, and Mark Shellenbaum. 2003. The zettabyte file system. In *Proceedings of the Usenix Conference on File and Storage Technologies (FAST)*, Vol. 215.

[5] Broadcom RAID Controller Cards. [n.d.]. . https://www.broadcom.com/products/storage/raid-controllers.

[6] Michael Burrows, Charles Jerian, Butler Lampson, and Timothy Mann. 1992. On-line data compression in a log-structured file system. *ACM SIGPLAN Notices* 27, 9 (1992), 2–9.

[7] Idan Burstein. 2021. Nvidia Data Center Processing Unit (DPU) Architecture. In *IEEE Hot Chips Symposium (HCS)*. 1–20. https://doi.org/10.1109/HCS52781.2021.9567066

[8] Canterbury Corpus. [n.d.]. . https://corpus.canterbury.ac.nz.

[9] Helen HW Chan, Yongkun Li, Patrick PC Lee, and Yinlong Xu. 2018. Elastic Parity Logging for SSD RAID Arrays: Design, Analysis, and Implementation. *IEEE Transactions on Parallel and Distributed Systems* 29, 10 (2018), 2241–2253.

[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 *Proceedings of USENIX Conference on File and Storage Technologies (FAST)*, Vol. 11. 77–90.

[11] Peter M Chen, Edward K Lee, Garth A Gibson, Randy H Katz, and David A Patterson. 1994. RAID: High-performance, reliable secondary storage. *ACM Computing Surveys (CSUR)* 26, 2 (1994), 145–185.

[12] Xubin Chen, Yin Li, Jingpeng Hao, Hyunsuk Shin, Michael Suh, and Tong Zhang. 2019. Simultaneously reducing cost and improving performance of NVM-based block devices via transparent data compression. In *Proceedings of the International Symposium on Memory Systems*. 331–341.

[13] Xubin Chen, Ning Zheng, Shukun Xu, Yifan Qiao, Yang Liu, Jiangpeng Li, and Tong Zhang. 2021. KallaxDB: A Table-less Hash-based Key-Value Store on Storage Hardware with Built-in Transparent Compression. In *Proceedings of the International Workshop on Data Management on New Hardware (DaMoN)*. 1–10.

[14] Derek Chiou, Eric Chung, and Susan Carrie. 2019. (Cloud) Acceleration at Microsoft. *Tutorial at Hot Chips* (2019).

[15] Dhananjoy Das, Dulcardo Arteaga, Nisha Talagala, Torben Mathiasen, and Jan Lindström. 2014. NVM Compression-Hybrid Flash-Aware Application Level Compression.. In *Workshop on Interactions of NVM/Flash with Operating Systems and Workloads (INFLOW)*.

[16] Dell EMC PowerMax. [n.d.]. . https://delltechnologies.com/.

[17] Jon Elerath and Michael Pecht. 2008. A highly accurate method for assessing reliability of redundant arrays of inexpensive disks (RAID). *IEEE Trans. Comput.* 58, 3 (2008), 289–299.

[18] Jon G Elerath and Jiri Schindler. 2014. Beyond MTTDL: A closed-form RAID 6 reliability equation. *ACM Transactions on Storage (TOS)* 10, 2 (2014), 1–21.

[19] Flexible I/O Tester. [n.d.]. . https://github.com/axboe/fio.

[20] E. F. Haratsch. 2019. SSD with Compression: Implementation, Interface and Use Case. In *Flash Memory Summit*.

[21] Mark Holland and Garth A Gibson. 1992. Parity declustering for continuous operation in redundant disk arrays. *ACM SIGPLAN Notices* 27, 9 (1992), 23–35.

[22] HPE Nimble Storage. [n.d.]. . https://www.hpe.com/.

[23] Jian Huang, Anirudh Badam, Moinuddin K Qureshi, and Karsten Schwan. 2015. Unified address translation for memory-mapped SSDs with FlashMap. In *Proceedings of the International Symposium on Computer Architecture (ISCA)*. 580–591.

[24] IBM Storwize. [n.d.]. . https://www.ibm.com/.

[25] Soojun Im and Dongkun Shin. 2010. Flash-aware RAID techniques for dependable and high-performance flash memory SSD. *IEEE Trans. Comput.* 60, 1 (2010), 80–92.

[26] William K Josephson, Lars A Bongo, Kai Li, and David Flynn. 2010. DFS: A file system for virtualized flash storage. *ACM Transactions on Storage (TOS)* 6, 3 (2010), 1–25.

[27] Yannis Klonatos, Thanos Makatos, Manolis Marazakis, Michail D Flouris, and Angelos Bilas. 2012. Transparent online storage compression at the block-level. *ACM Transactions on Storage (TOS)* 8, 2 (2012), 1–33.

[28] Yongkun Li, Patrick PC Lee, and John CS Lui. 2013. Stochastic analysis on RAID reliability for solid-state drives. In *IEEE International Symposium on Reliable Distributed Systems*. 71–80.

[29] Linux mdraid. [n.d.]. . https://raid.wiki.kernel.org/.

[30] Fumio Machida, Ruofan Xia, and Kishor S Trivedi. 2015. Performability modeling for RAID storage systems by Markov regenerative process. *IEEE Transactions on Dependable and Secure Computing* 15, 1 (2015), 138–150.

[31] Sangwhan Moon and AL Narasimha Reddy. 2013. Don't Let RAID Raid the Lifetime of Your SSD Array. In *USENIX Workshop on Hot Topics in Storage and File Systems (HotStorage)*.

[32] Richard R. Muntz and John C. S. Lui. 1990. Performance Analysis of Disk Arrays under Failure. In *International Conference on Very Large Data Bases (VLDB)*. 162–173.

[33] N. Muppalaneni and K. Gopinath. 2000. A multi-tier RAID storage system with RAID1 and RAID5. In *Proceedings of International Parallel and Distributed Processing Symposium (IPDPS)*. 663–671. https://doi.org/10.1109/IPDPS.2000.846051

[34] David A Patterson, Garth Gibson, and Randy H Katz. 1988. A case for redundant arrays of inexpensive disks (RAID). In *Proceedings of the ACM international conference on Management of data (SIGMOD)*. 109–116.

[35] Pure Storage FlashBlade. [n.d.]. . https://www.purestorage.com/.

[36] Yifan Qiao, Xubin Chen, Ning Zheng, Jiangpeng Li, Yang Liu, and Tong Zhang. 2022. Closing the B+-tree vs. LSM-tree Write Amplification Gap on Modern Storage Hardware with Built-in Transparent Compression. In *USENIX Conference on File and Storage Technologies (FAST)*. 1–14.

[37] RAIDIX. [n.d.]. . https://www.raidix.com.

[38] Ohad Rodeh, Josef Bacik, and Chris Mason. 2013. BTRFS: The Linux B-tree filesystem. *ACM Transactions on Storage (TOS)* 9, 3 (2013), 1–32.

[39] Mohit Saxena, Michael M Swift, and Yiying Zhang. 2012. FlashTier: a lightweight, consistent and durable storage cache. In *Proceedings of the ACM European conference on Computer Systems*. 267–280.

[40] ScaleFlux Computational Storage. [n.d.]. . http://scaleflux.com.

[41] Abraham Silberschatz, Peter B Galvin, and Greg Gagne. 2013. *Operating system concepts essentials*. Wiley Publishing.

[42] Silesia Corpus. [n.d.]. . https://github.com/MiloszKrajewski/SilesiaCorpus.

[43] Kiran Srinivasan, Timothy Bisson, Garth R Goodson, and Kaladhar Voruganti. 2012. iDedup: latency-aware, inline data deduplication for primary storage.. In *USENIX Conference on File and Storage Technologies (FAST)*. 1–14.

[44] Alexander Thomasian and Mario Blaum. 2009. Higher reliability redundant disk arrays: Organization, operation, and coding. *ACM Transactions on Storage (TOS)* 5, 3 (2009), 1–59.

[45] Derek Vadala. 2002. *Managing RAID on Linux: Fast, Scalable, Reliable Data Storage*. O'Reilly Media, Inc.

[46] Virtual Data Optimizer (VDO). [n.d.]. . https://github.com/dm-vdo/vdo.

[47] Jiguang Wan, Jibin Wang, Qing Yang, and Changsheng Xie. 2010. S2-RAID: A new RAID architecture for fast data recovery. In *IEEE Symposium on Mass Storage Systems and Technologies (MSST)*. 1–9.

[48] Yan Wang, Xunrui Yin, and Xin Wang. 2014. MDR codes: A new class of RAID-6 codes with optimal rebuilding and encoding. *IEEE Journal on Selected Areas in Communications* 32, 5 (2014), 1008–1018.

[49] John Wilkes, Richard Golding, Carl Staelin, and Tim Sullivan. 1996. The HP AutoRAID hierarchical storage system. *ACM Transactions on Computer Systems (TOCS)* 14, 1 (1996), 108–136.

[50] Guanying Wu and Xubin He. 2012. Delta-FTL: improving SSD lifetime via exploiting content locality. In *Proceedings of the 7th ACM European Conference on Computer Systems (EuroSys)*. 253–266.

[51] Liping Xiang, Yinlong Xu, John CS Lui, and Qian Chang. 2010. Optimal recovery of single disk failure in RDP code storage systems. *ACM SIGMETRICS Performance Evaluation Review* 38, 1 (2010), 119–130.

[52] zlib. [n.d.]. . http://zlib.net.

[53] Aviad Zuck, Sivan Toledo, Dmitry Sotnikov, and Danny Harnik. 2014. Compression and SSDs: Where and how?. In *Workshop on Interactions of NVM/Flash with Operating Systems and Workloads (INFLOW)*.