Asymmetric RAID: Rethinking RAID for SSD Heterogeneity

Ziyang Jiao
Syracuse University
zjiao04@syr.edu

Bryan S. Kim
Syracuse University
bkim01@syr.edu

Abstract

Traditional RAID solutions (e.g., Linux MD) balance writes evenly across the array for high I/O parallelism and data reliability. This is built around the assumption that the underlying storage components are homogeneous, both in performance and capacity. However, SSDs, even for the same model, exhibit very different characteristics and degrade over time, leading to severe disk under-utilization.

In this work, we present Asymmetric-RAID (Asym-RAID), a novel RAID architecture that optimizes system performance and storage utilization by exploiting heterogeneity from a larger SSD pool. Asym-RAID asymmetrically distributes data across the array to fully utilize the capacity of each SSD. To improve performance, Asym-RAID differentially exports the address space of each data stripe to the host, allowing for performance-optimized data placement. We outline the necessary changes in the storage stack for building an asymmetric RAID system and highlight its benefits.

CCS Concepts: • **Information systems** → **Disk arrays**; **RAID**; **Flash memory**; • **Computing methodologies** → **Modeling methodologies**.

Keywords: SSD, RAID, All-flash array, performance, capacity, modeling, heterogeneity

ACM Reference Format:
Ziyang Jiao and Bryan S. Kim. 2024. Asymmetric RAID: Rethinking RAID for SSD Heterogeneity. In *16th ACM Workshop on Hot Topics in Storage and File Systems (HOTSTORAGE '24), July 8–9, 2024, Santa Clara, CA, USA.* ACM, New York, NY, USA, 7 pages. https://doi.org/10.1145/3655038.3665952

1 Introduction

The explosive growth of data, driven by the era of big data, places significant stress on data center storage systems [17]. All-flash arrays (AFAs), which leverage the high performance, low power consumption, and compact form factor of solid-state drives (SSDs), are seen as a promising solution to meet the growing demand for storage [8, 17].

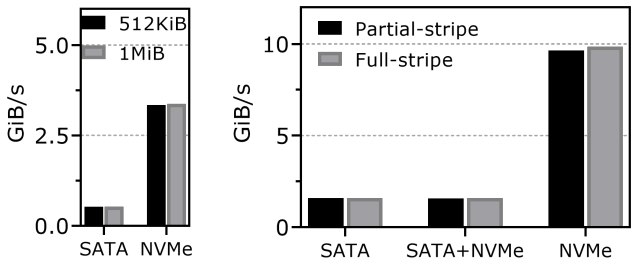
However, despite providing numerous advantages, SSDs present unique challenges compared to conventional hard disk drives (HDDs) [5, 10, 11, 22, 29]. The performance and capacity of SSDs can vary significantly depending on the form factor and flash memory chips, particularly for enterprise-grade products [32]. Moreover, it is not uncommon for the performance of a modern SSD to degrade over time due to various factors such as write amplification and flash memory errors, leading to "fail-slow" symptoms where the components continue to function but experience reduced performance [5, 10, 29]. Recent studies have demonstrated that fail-slow drives can cause latency spikes of up to 3.65× [22] in large-scale storage clusters and the performance of a modern SSD degrades at a rate of 4.3% per petabyte data written [11].

On the other hand, the overall architecture of existing AFA systems is built around the assumption that the underlying storage components are homogeneous. A common N-disk RAID maintains a deterministic stripes-to-disk mapping that uniformly distributes data and parity chunks across all disks [38]. This architecture results in significant disk under-utilization when considering heterogeneity among storage devices, particularly for modern SSDs. The balanced data layout requires the underlying storage devices to be roughly the same size. Otherwise, the aggregate capacity is determined by the minimal capacity device, making devices with larger capacity underutilized. Moreover, by evenly spreading data across the disks, the overall system performance is bottlenecked by the poor-performing drives.

We illustrate this by testing the popular Linux-MD software RAID-5 array built on heterogeneous SSDs. Figure 1a shows the performance profile of two types of SSD products used for the experiments, measured by fio [7]. The read throughput of the NVMe SSD outperforms the SATA SSD by 5.31× and 5.35× with 512 KiB and 1 MiB request sizes, respectively. We then compare three RAID-5 systems: (1) SATA: consisting of 3 SATA SSDs; (2) SATA+NVMe: consisting of 1 SATA SSD + 2 NVMe SSDs; and (3) NVMe: consisting of 3 NVMe SSDs. As shown in Figure 1b, although utilizing two NVMe SSDs, the performance of the SATA+NVMe RAID remains similar to the SATA RAID and achieves only 16.2% and 16.1% of the NVMe RAID performance, for partial-stripe

HOTSTORAGE '24, July 8–9, 2024, Santa Clara, CA, USA

ACM ISBN 979-8-4007-0630-1/24/07
https://doi.org/10.1145/3655038.3665952



(a) Performance comparison of SATA and NVMe SSDs under sequential read.

(b) Performance comparison of three RAID-5 systems. SATA (3 SATA SSDs) vs. SATA+NVMe (1 SATA SSD and 2 NVMe SSDs) vs. NVMe (3 NVMe SSDs).

Figure 1. Performance bottleneck caused by disk heterogeneity with Linux MD. Figure 1a shows the read performance of a SATA and an NVMe SSD. Figure 1b demonstrates that the overall RAID performance is determined by the device with the lowest performance with conventional RAID solutions.

read and full-stripe read, respectively. As a result, with the conventional RAID solution, the performance is determined by the lowest-performance device, resulting in significant under-utilization of system resources.

As the latest Linux MD is capable of supporting arrays with more than 384 component devices [21], large cluster storage systems almost always include a heterogeneous mix of storage devices [12]. For example, RAID groups from NetApp deployed in the field contain SSDs with varying deployment times, which presents different performance characteristics [23]. Storage nodes deployed in Alibaba Cloud employ 12 to 18 SSDs from multiple vendors to support cloud services, including Block service, NoSQL service, and Big Data service [37]. Furthermore, Huawei builds storage clusters consisting of different types of storage devices, including HDDs and SSDs, to improve utilization and throughput [6]. Consequently, such heterogeneous environments inevitably lead to utilization issues and performance penalties with the state-of-the-art RAID architectures [12].

This paper presents Asymmetric-RAID, a new RAID construction mechanism to address disk heterogeneity for modern SSDs. Asym-RAID is designed to asymmetrically distribute data across the array to fully utilize the capacity of each SSD while mathematically guaranteeing a maximum logical volume exported to the host. To avoid performance bottlenecks, Asym-RAID maintains multiple stripe groups based on their performance characteristics and differentially exports the address space of each data stripe to the host, allowing a more performance-optimized data placement.

The contributions of this paper are as follows.

- We analyze and demonstrate the inefficiency of conventional RAID solutions when considering disk heterogeneity for modern SSDs.
- We present Asymmetric-RAID, to our knowledge, the first RAID architecture designed to leverage a mix of heterogeneous SSDs to improve overall storage utilization and efficiency by distributing data asymmetrically.
- We investigate the option of building a performance-aware logical volume by exporting address spaces differentially and coordinating with file systems.

The rest of the paper is organized as follows. Section 2 presents the background for RAID and illustrates the performance changes of modern SSDs. Section 3 describes Asym-RAID's design, including its heterogeneity-aware data distribution and performance-optimized data placement. We then discuss other relevant issues in Section 4 and conclude by highlighting the heterogeneity issue in RAID environments and the need for further research attention in Section 5.

2 Background and Related Works

In this section, we first present the background for RAID. We then demonstrate that performance heterogeneity is a common issue in SSDs, even among devices of the same model. Finally, we review existing approaches for managing and optimizing AFA storage.

2.1 RAID

Redundant array of independent disks (RAID) is a classic solution that manages an array of SSDs to improve performance, reliability, and capacity simultaneously [36]. In RAID, data is striped across the drives based on the required level of redundancy and performance, referred to as RAID levels.

RAID-5 and RAID-6 are the most commonly used parity-based RAID levels to balance throughput, redundancy, and space overhead [34]. For an N-drive RAID-5 array, each data stripe consists of $N-1$ data chunks and one parity chunk, which rotates for each stripe to eliminate any parity bottlenecks in the array [1]. As a result, RAID-5 offers redundancy against one drive failure with $1/(N-1)$ space overhead. In theory, RAID-5 can deliver read throughput that scales up to N times and write throughput that scales up to $N-1$ times compared to the throughput of a single device [34].

However, in practice, this throughput can hardly be achieved even *without* drive failures [8, 19, 38]. There are three write modes in parity-based RAID: full-stripe write, read-modify-write, and reconstruct write. For a partial-stripe write (read-modify-write and reconstruct write) where only a subset of the data chunks of a stripe is written, parity chunks or old data chunks need to be read from corresponding drives to generate new parity chunks, thus amplifying the amount of I/Os. In terms of read, achieving ideal performance relies on the assumption that the underlying drives can simultaneously deliver consistent performance. However, this assumption does not hold true in practice, particularly with modern SSDs. We will discuss this impact in the subsequent section.

2.2 SSD performance heterogeneity

Based on the design methodology of manufacturing, modern SSDs composed of persistent memory (PM), triple-level cell (TLC), or quadruple-level cell (QLC) flash memory can exhibit very different performance characteristics. A recent study has shown that the performance differential between two commodity SSD devices can be as high as 5.1× [36]. Thus, to enhance overall resource utilization, it is often recommended to construct RAID systems using disks of the same make and model [12, 20].

Nevertheless, the challenge of heterogeneous performance persists even among disks of identical models [1, 5, 9, 11, 15, 15, 33]. First, SSDs inherently exhibit performance variability from the time of manufacturing due to differences in the quality of NAND flash memory cells and variations in the firmware [2, 33]. Second, depending on workloads, SSDs from the same brand can experience varying levels of degradation within RAID configurations. For instance, with partial-stripe writes, only devices containing relevant data and parity chunks are updated while the remaining data chunks within the stripe are untouched. As the SSDs receive write requests, flash memories degrade differently due to wear [1, 9, 15, 28] and disturbance [30]. Previous studies have demonstrated the fail-slow symptoms in SSDs, and they can exhibit very different performance characteristics over time [5, 11, 15].

We illustrate this by conducting experiments on an enterprise-grade SSD and measuring its performance under different conditions. The device is used by performing random writes, with approximately 100 terabytes of data written each day over a duration of 90 days. We measure the latency distribution under read-only I/O workloads (i.e., 4 KiB sequential read) every morning when the server is idle to avoid other impacts such as SSD garbage collection and host disruptions. As shown in Figure 2, even without the impact of GC and host, the device fails to deliver consistent performance over time. The average latency increases by 96% compared to its initial state after running the same workload for 90 days. As NAND flash memory density continues to scale, we expect the degree of such performance changes to increase [11].

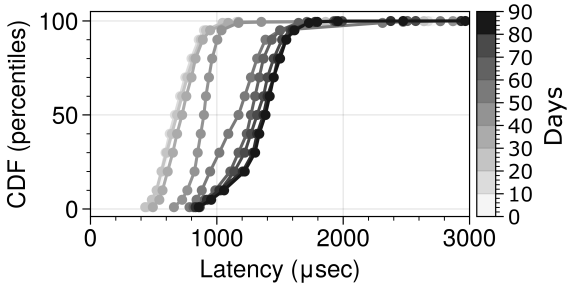


Figure 2. Latency distributions of *read-only* workloads on an enterprise-grade SSD under its different states. The device is used through random writes each day. Even without the impact of GC and host disruptions, the device presents different performance characteristics over time.

2.3 All-flash array systems

As summarized in Table 1, AFA systems have been extensively studied, with approaches categorized into three groups: (1) taming tail-latency by alleviating GC impact [17, 19]; (2) enhancing performance by reducing software overhead [8, 36, 38]; (3) improving reliability by distributing parity unevenly across the devices [1] or adopting disk-adaptive data redundancy scheme [12–14]. Approaches in the first two groups typically assume that disk components have the same capacity and comparable performance, and most in the third group only focus on the reliability perspective of the system.

Alleviating GC interference. SWAN [17] organizes SSDs into foreground and background groups. The foreground group will be used to accommodate host writes and only SSDs in background groups are allowed to perform GC. It aims to match the aggregated group bandwidth with the network performance without considering disk heterogeneity. IODA [19], on the other hand, proposes a busy time model to

Table 1. Comparison of existing approaches when managing heterogeneous SSDs in All-Flash Array storage.

	Disk heterogeneity	Device profiling reliance	Metadata overhead	Deployment cost	Disk utilization
Linux-MD [25]	Low	–	Low	Low	Low
SWAN (Log-RAID) [17]	Low	Low	High	Medium	Medium
IODA [19]	Low	Medium	Low	High	Medium
RAID+ [38]	Low	–	Low	Low	Medium
FusionRAID [8]	Low	High	Medium	High	Medium
StRAID [36]	Low	–	Low	Low	Medium
Diff-RAID [1]	Low	Low	Medium	Medium	Low
HeART [14]	Medium	Medium	High	High	Low
Pacemaker [13]	Medium	High	Medium	High	Low
Tiger [12]	Medium	High	Medium	Medium	Medium
Asym-RAID (proposed)	High	Low	Low	Low	High

coordinate GC and achieve I/O determinism. The model considers devices to have the same capacity and performance.

Reducing software overhead. Both RAID+ [38] and FusionRAID [8] utilize the mathematical properties of mutually orthogonal Latin squares (MOLS) [38] to spread I/O to a larger disk pool in a balanced manner. Unfortunately, MOLS requires each device to have exactly the same size [24]. FusionRAID relies on latency spike detection and requests redirection to avoid drives experiencing degraded performance, causing additional system overhead. Moreover, to reduce mapping overhead, it restricts a relatively large block size. On the other hand, StRAID [36] addresses the lock contentions in Linux-MD [25] by assigning a dedicated thread for each stripe write. It also adopts a two-phase write mechanism to opportunistically aggregate I/Os.

Improving system reliability. Diff-RAID [1] distributes parity blocks unevenly across the array and maintains an age differential to lower correlated failures in SSDs. However, it concerns the case where the array has operated in the degraded mode — data can not be reconstructed because of correlated failures once a drive fails.

HeART [14] and Pacemaker [13] propose disk-adaptive redundancy schemes to ensure data reliability. To address the I/O overhead from redundancy transitions [13], Pacemaker proactively makes predictions to spread the transition overload out over time. These two approaches contain disks from only one make/model. Similarly, Tiger [12] tailors the redundancy scheme of each data stripe based on disks' annualized failure rates (AFRs). Works within this group focus on addressing disk heterogeneity in terms of reliability rather than capacity and performance.

3 Asymmetric RAID

Inspired by the above observations and analysis, we introduce Asymmetric-RAID. In this section, we first present the overview and challenges of the proposed design. We then describe the methodology for data distribution and the performance-optimized data placement scheme.

3.1 Overview

The proposed Asymmetric-RAID is designed to be heterogeneity-aware. The core idea is to asymmetrically distribute data across a larger disk pool based on each device's capacity and performance. By doing so, more data will be placed in larger SSDs and less in smaller ones. Figure 3 illustrates the overall architecture of the Asym-RAID, for a simple (2+1) RAID-5 configuration (i.e., 2 data chunks and 1 parity chunk) from a 5-disk array. Asym-RAID introduces a two-dimensional logical address space, an internal logical block layer between the user-perceived logical and the SSD logical address spaces, where each row corresponds to a single device. Asym-RAID then separates each address space into one or more stripe groups and differentially exports them into a one-dimensional logical address space to the host based on the device performance.

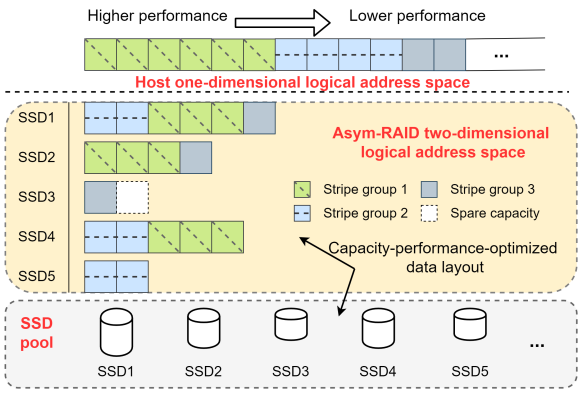


Figure 3. Overview of Asym-RAID: a simple (2+1) RAID-5 configuration from a 5-disk array.

The novel challenges here are to (1) design an efficient data organization to utilize the total aggregate capacity given a disk array where each device has a different capacity; (2) achieve address translation between user space, AFA space, and devices with minimal overhead; as well as (3) develop a simple scheme to enable the host to optimize the usage of devices with higher performance. We first explain the data distribution scheme that stripes data unevenly across devices and then describe the capacity-performance-optimized data placement enabled by the Asym-RAID.

3.2 Heterogeneity-aware data distribution

For conventional N-disk RAID arrays, each data stripe consists of exactly N chunks, which are uniformly distributed across N disks. To distribute data across a disk pool that exceeds the typical size of RAID arrays, one straightforward approach is to physically partition the disk pool into two RAID groups [38], each potentially configured with different RAID settings, where each disk belongs to one RAID array. RAID-50, for instance, adheres to this approach. However, while this approach achieves good disk isolation and efficient stripes-to-disks mapping, only a fixed amount of capacity is used for each device.

Alternatively, Aysm-RAID unevenly distributes data across the SSDs, with the goal of fully utilizing the capacity of each SSD and maximizing the available logical capacity exported to the host. Consequently, SSDs with larger capacity will be assigned more data and parity chunks within the array.

In particular, this can be formulated as an optimization problem as follows: given disk pool size N, size of the i^{th} disk S_i (where i ranges from 1 to N), data stripe width k ($k < N$), and chunk size C. Let x_{ijk} be a binary decision variable representing whether chunk k of data stripe j is assigned to disk i. The objective function is to maximize the number of complete k-width data stripes, denoted by

D. There are three constraints inherited from the RAID to ensure data reliability: 1) each chunk of each data stripe must be assigned to exactly one disk; 2) any two chunks within a data stripe are placed on different disks; and 3) each disk can only accommodate a certain number of chunks based on its size. Therefore, we have:

$$\begin{aligned} &\text{Maximize} \quad D \\ &\sum_{i=1}^{N} x_{ijk} = 1, \quad \forall j, \forall k \\ &x_{ijk} + x_{ijk'} \leq 1, \quad \forall i, \forall j, \forall k' \neq k \\ &\sum_{j=1}^{D} \sum_{k=1}^{k} C \cdot x_{ijk} \leq S_i, \quad \forall i \end{aligned} \tag{1}$$

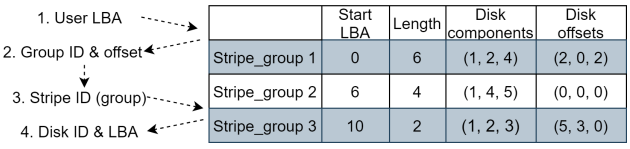
This combinatorial optimization problem can be commonly solved by taking advantage of integer linear programming (ILP) techniques [31] and Figure 3 presents a simple example of the derived stripe organization, with $N = 5$, $k = 3$, $S = \{6, 4, 2, 5, 2\}$, and $C = 1$. The derived stripe organization mathematically guarantees a maximum logical address space given a disk array while allowing Asym-RAID to asymmetrically place data to fully utilize the aggregate capacity.

The derived stripe organization has another property that reduces metadata overhead for stripes-to-disk mapping: each k-subset of n disks can belong to at most one stripe group (i.e., all stripes from those k disks). Asym-RAID designs a stripe state table (SST) to maintain the mapping information of each stripe group, which converts a user LBA to a device LBA. Figure 4 presents an example of the SST corresponding to the data layout shown in Figure 3, assuming the chunk/block size is 1. Specifically, an SST entry contains four fields for the indexed stripe group: `start_LBA`, `length`, `disk_components`, which determines the relative order of data and parity chunks among the involved disks, and `disk_offsets`, which indicates the starting disk LBA for this stripe group.

3.3 Performance-optimized placement

So far, Asym-RAID employs a deterministic mapping to translate an upper-level LBA to an SSD logical address by consulting the SST table, which improves capacity utilization and reduces the metadata overhead for addressing compared to dynamic mapping mechanisms [8, 12, 17]. However, there is another issue with how to integrate the address space of each stripe group into a single logical volume and export it to the upper layers (i.e., file systems and block I/O layer). Simply concatenating them would not yield performance benefits, as logical blocks are equally treated and accessed from the perspective of the file system.

To tackle this, Asym-RAID first profiles the performance characteristics of every stripe group, which involves fixed disk components derived from § 3.2. Asym-RAID then concatenates and exports the address space of each stripe group



	Start LBA	Length	Disk components	Disk offsets
Stripe_group 1	0	6	(1, 2, 4)	(2, 0, 2)
Stripe_group 2	6	4	(1, 4, 5)	(0, 0, 0)
Stripe_group 3	10	2	(1, 2, 3)	(5, 3, 0)

Figure 4. Asym-RAID block mapping table. The stripes-to-disk mapping is determined during RAID initialization.

based on the identified performance. As a result, the LBA0 perceived by the user will be mapped to the most performant disks, while subsequent LBAs will be allocated to progressively less performant disks. As illustrated in Figure 3, the green blocks correspond to the most performant stripe group 0, which consists of SSD 1, 2, and 4. The subsequent blue and gray blocks are mapped to less performant disk components.

In this example, the difference in terms of disk components between stripe group 1 and stripe group 2 is SSD 2 and SSD 5, which suggests that the performance bottleneck of stripe group 2 is attributed to SSD 5. This is because the data distribution algorithm described in § 3.2 is more optimized for maximizing the aggregate logical capacity. One approach to address this is to first partition the array into disk groups with comparable performance and then apply the data distribution algorithm within these groups. It alleviates performance disparities among disks within the same group at the cost of decreased logical capacity.

With that, it opens the door for the system to differentially use logical blocks with negligible overhead, leading to disks with higher performance being better utilized. For example, F2FS internally divides the address space into metadata area and data area, which is further divided into several (i.e., six by default) hot/cold logs [18]. As a result, performance-sensitive data (i.e., metadata and hot logs) can be placed in the lower LBA space for optimized performance, and cold data can be placed in the higher LBA space for better capacity utilization.

4 Discussion and Future Work

We now discuss related issues and future work, including implementation challenges, dynamic heterogeneity, disk replacement, fault tolerance, and using Aysm-RAID over disaggregated storage.

Implementation challenges. In conventional RAID solutions, physical addresses can be directly computed with minimal overhead due to the predefined data layout. In contrast, Asym-RAID requires a logical-to-physical mapping for each stripe group, which takes up to 25 bytes. Additionally, the entire SST table must be persisted in the RAID superblock, leading to extra space and performance overhead. Nevertheless, the metadata overhead introduced by Asym-RAID does not exceed that of Log-RAID [3, 17], which maintains a one-to-one mapping for each logical block/chunk. Furthermore,

the addressing overhead can be mitigated with techniques such as learned index models [4, 35].

Dynamic heterogeneity. The proposed architecture addresses the static disk heterogeneity when the system is initialized. However, as described in § 2.2, the performance of SSDs can gradually degrade throughout their lifetime, which needs automatically optimizing data layout to adapt to dynamic disk heterogeneity. The challenge here is to minimize the data re-distribution overhead. One way to solve this problem is via data-remapping [27, 39]. However, we believe that this approach may not be optimal as in RAID, data is located on different disks.

Disk replacement. In Asym-RAID, each drive can contribute to multiple stripe groups. Different from conventional RAID, when a larger capacity device is used, the added capacity can potentially combine with spare capacity from other disks to create new data stripes. By inserting them into existing stripe groups or developing new ones, these data stripes extend the storage capacity. We consider investigating recent works [11, 16] to coordinate Asym-RAID and file systems for efficient online address space adjustment.

Reliability and fault tolerance. At the current stage, Asym-RAID operates in the same way as the legacy MD under disk failures. The missing data needs to be reconstructed from the remaining disks in the array. We plan to consider the case where the reliability of the SSDs is also different. Unlike existing disk-adaptive redundancy schemes [12, 13], we will exploit machine-learning techniques to predict SSD reliability changes and imbue this information into the block I/O layer for more efficient data management and fault handling.

RAID over disaggregated storage. Asym-RAID can be deployed with disaggregated storage using the NVMe-over-Fabrics (NVMe-oF) protocol [26]. However, by aggregating a few SSDs, the storage bandwidth can easily surpass the bandwidth of a high-end RDMA NIC (e.g., 100 Gbps) [17, 34], and thus it is still challenging to unlock the full performance of modern SSDs. One approach is to leverage SSD internal hardware resources and adopt a host/device co-design that enables them to communicate directly with their peers without host intervention.

5 Conclusion

From a system design perspective, it is easier to construct and manage RAID with a deterministic disk layout that evenly distributes I/O across the disk pool. However, with advancements in SSD technology and evolving data demands, the limitations of traditional RAID systems are becoming increasingly evident, particularly in large cluster storage systems where heterogeneous storage devices are ubiquitous. We believe that it is imperative to rethink existing RAID architectures and leverage SSD heterogeneity to fully utilize the capabilities of each storage device.

Acknowledgments

We thank the anonymous reviewers for their constructive comments and insightful suggestions that help us to improve the quality of this paper. This research was supported, in part, by the National Science Foundation award CNS-2008453.

References

[1] Mahesh Balakrishnan, Asim Kadav, Vijayan Prabhakaran, and Dahlia Malkhi. 2010. Differential RAID: rethinking RAID for SSD reliability. In *European Conference on Computer Systems (EuroSys'10)*. ACM, 15–26.

[2] Feng Chen, David A. Koufaty, and Xiaodong Zhang. 2009. Understanding intrinsic characteristics and system implications of flash memory based solid state drives. In *International Conference on Measurement and Modeling of Computer Systems (SIGMETRICS'09)*. ACM, 181–192.

[3] Tzi-cker Chiueh, Weafon Tsao, Hou-Chiang Sun, Ting-Fang Chien, An-Nan Chang, and Cheng-Ding Chen. 2014. Software Orchestrated Flash Array. In *International Systems and Storage Conference (SYSTOR'14)*. ACM, 14:1–14:11.

[4] Jialin Ding, Umar Farooq Minhas, Jia Yu, Chi Wang, Jaeyoung Do, Yinan Li, Hantian Zhang, Badrish Chandramouli, Johannes Gehrke, Donald Kossmann, David B. Lomet, and Tim Kraska. 2020. ALEX: An Updatable Adaptive Learned Index. In *International Conference on Management of Data (SIGMOD'20)*. ACM, 969–984.

[5] Haryadi S. Gunawi, Riza O. Suminto, Russell Sears, Casey Golliher, Swaminathan Sundararaman, Xing Lin, Tim Emami, Weiguang Sheng, Nematollah Bidokhti, Caitie McCaffrey, Gary Grider, Parks M. Fields, Kevin Harms, Robert B. Ross, Andree Jacobson, Robert Ricci, Kirk Webb, Peter Alvaro, H. Birali Runesha, Mingzhe Hao, and Huaicheng Li. 2018. Fail-Slow at Scale: Evidence of Hardware Performance Faults in Large Production Systems. In *Conference on File and Storage Technologies (FAST'18)*. USENIX Association, 1–14.

[6] Huawei. 2024. Huawei Data Storage. https://e.huawei.com/en/products/storage.

[7] Jens Axboe. 2005. Flexible I/O tester. https://github.com/axboe/fio/.

[8] 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 *Conference on File and Storage Technologies (FAST'21)*. USENIX Association, 355–370.

[9] Ziyang Jiao, Janki Bhimani, and Bryan S. Kim. 2022. Wear leveling in SSDs considered harmful. In *Workshop on Hot Topics in Storage and File Systems (HotStorage'22)*. ACM, 72–78.

[10] Ziyang Jiao and Bryan S. Kim. 2022. Generating realistic wear distributions for SSDs. In *Workshop on Hot Topics in Storage and File Systems (HotStorage'22)*. ACM, 65–71.

[11] Ziyang Jiao, Xiangqun Zhang, Hojin Shin, Jongmoo Choi, and Bryan S. Kim. 2024. The Design and Implementation of a Capacity-Variant Storage System. In *Conference on File and Storage Technologies (FAST'24)*. USENIX Association, 159–176.

[12] Saurabh Kadekodi, Francisco Maturana, Sanjith Athlur, Arif Merchant, K. V. Rashmi, and Gregory R. Ganger. 2022. Tiger: Disk-Adaptive Redundancy Without Placement Restrictions. In *Symposium on Operating Systems Design and Implementation (OSDI'22)*. USENIX Association, 413–429.

[13] Saurabh Kadekodi, Francisco Maturana, Suhas Jayaram Subramanya, Juncheng Yang, K. V. Rashmi, and Gregory R. Ganger. 2020. PACEMAKER: Avoiding HeART attacks in storage clusters with disk-adaptive redundancy. In *Symposium on Operating Systems Design and Implementation (OSDI'20)*. USENIX Association, 369–385.

[14] Saurabh Kadekodi, K. V. Rashmi, and Gregory R. Ganger. 2019. Cluster storage systems gotta have HeART: improving storage efficiency by exploiting disk-reliability heterogeneity. In *Conference on File and Storage Technologies (FAST'19)*. USENIX Association, 345–358.

[15] Bryan S. Kim, Jongmoo Choi, and Sang Lyul Min. 2019. Design Tradeoffs for SSD Reliability. In *Conference on File and Storage Technologies (FAST'19)*. USENIX Association, 281–294.

[16] Juwon Kim, Minsu Kim, Muhammad Danish Tehseen, Joontaek Oh, and Youjip Won. 2022. IPLFS: Log-Structured File System without Garbage Collection. In *Annual Technical Conference (ATC'22)*. USENIX Association, 739–754.

[17] Jaeho Kim, Kwanghyun Lim, Youngdon Jung, Sungjin Lee, Changwoo Min, and Sam H. Noh. 2019. Alleviating Garbage Collection Interference Through Spatial Separation in All Flash Arrays. In *Annual Technical Conference (ATC'19)*. USENIX Association, 799–812.

[18] Changman Lee, Dongho Sim, Joo Young Hwang, and Sangyeun Cho. 2015. F2FS: A New File System for Flash Storage. In *Conference on File and Storage Technologies (FAST'15)*. USENIX Association, 273–286.

[19] Huaicheng Li, Martin L. Putra, Ronald Shi, Xing Lin, Gregory R. Ganger, and Haryadi S. Gunawi. 2021. IODA: A Host/Device Co-Design for Strong Predictability Contract on Modern Flash Storage. In *Symposium on Operating Systems Principles (SOSP'21)*. ACM, 263–279.

[20] Linux RAID. 2011. Linux RAID Setup. https://raid.wiki.kernel.org/index.php/RAID_setup#RAID-4.2F5.2F6.

[21] Linux RAID. 2011. Linux RAID Superblock. https://raid.wiki.kernel.org/index.php/RAID_superblock_formats.

[22] Ruiming Lu, Erci Xu, Yiming Zhang, Fengyi Zhu, Zhaosheng Zhu, Mengtian Wang, Zongpeng Zhu, Guangtao Xue, Jiwu Shu, Minglu Li, and Jiesheng Wu. 2023. Perseus: A Fail-Slow Detection Framework for Cloud Storage Systems. In *Conference on File and Storage Technologies (FAST'23)*. USENIX Association, 49–64.

[23] Stathis Maneas, Kaveh Mahdaviani, Tim Emami, and Bianca Schroeder. 2020. A Study of SSD Reliability in Large Scale Enterprise Storage Deployments. In *Conference on File and Storage Technologies (FAST'20)*. USENIX Association, 137–149.

[24] Henry B. Mann. 1942. The construction of orthogonal Latin squares. In *The Annals of Mathematical Statistics*. JSTOR, 418–423.

[25] NeilBrown. 2020. Multiple device driver. https://github.com/neilbrown/mdadm.

[26] NVM Express. 2021. NVMe over Fabrics. https://nvmexpress.org/wp-content/uploads/NVMe_Over_Fabrics.pdf.

[27] Gihwan Oh, Chiyoung Seo, Ravi Mayuram, Yang-Suk Kee, and Sang-Won Lee. 2016. SHARE Interface in Flash Storage for Relational and NoSQL Databases. In *International Conference on Management of Data (SIGMOD'16)*. ACM, 343–354.

[28] Open NAND Flash Interface. 2021. ONFI 5.0 Spec. http://www.onfi.org/specifications/.

[29] Biswaranjan Panda, Deepthi Srinivasan, Huan Ke, Karan Gupta, Vinayak Khot, and Haryadi S. Gunawi. 2019. IASO: A Fail-Slow Detection and Mitigation Framework for Distributed Storage Services. In *Annual Technical Conference (ATC'19)*. USENIX Association, 47–62.

[30] Jisung Park, Myungsuk Kim, Myoungjun Chun, Lois Orosa, Jihong Kim, and Onur Mutlu. 2021. Reducing solid-state drive read latency by optimizing read-retry. In *Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS'21)*. ACM, 702–716.

[31] Arvind Rajan. 1990. Theory of linear and integer programming. In *Networks*. Wiley, 801.

[32] Samsung. 2024. Samsung Datacenter SSDs. https://semiconductor.samsung.com/us/ssd/datacenter-ssd/pm9a3/mzql23t8hcls-00a07/.

[33] Seagate. 2021. BarraCuda 120 SSD Data Sheet. https://www.seagate.com/content/dam/seagate/migrated-assets/www-content/datasheets/pdfs/barracuda-120-sata-DS2022-2-2104US-en_US.pdf.

[34] Junyi Shu, Ruidong Zhu, Yun Ma, Gang Huang, Hong Mei, Xuanzhe Liu, and Xin Jin. 2023. Disaggregated RAID Storage in Modern Datacenters. In *Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS'23)*. ACM, 147–163.

[35] Jinghan Sun, Shaobo Li, Yunxin Sun, Chao Sun, Dejan Vucinic, and Jian Huang. 2023. LeaFTL: A Learning-Based Flash Translation Layer for Solid-State Drives. In *Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS'23)*. ACM, 442–456.

[36] Shucheng Wang, Qiang Cao, Ziyi Lu, Hong Jiang, Jie Yao, and Yuanyuan Dong. 2022. StRAID: Stripe-threaded Architecture for Parity-based RAIDs with Ultra-fast SSDs. In *Annual Technical Conference (ATC'22)*. USENIX Association, 915–932.

[37] Erci Xu, Mai Zheng, Feng Qin, Yikang Xu, and Jiesheng Wu. 2019. Lessons and Actions: What We Learned from 10K SSD-Related Storage System Failures. In *Annual Technical Conference (ATC'19)*. USENIX Association, 961–976.

[38] Guangyan Zhang, Zican Huang, Xiaosong Ma, Songlin Yang, Zhufan Wang, and Weimin Zheng. 2018. RAID+: Deterministic and Balanced Data Distribution for Large Disk Enclosures. In *Conference on File and Storage Technologies (FAST'18)*. USENIX Association, 279–294.

[39] You Zhou, Qiulin Wu, Fei Wu, Hong Jiang, Jian Zhou, and Changsheng Xie. 2021. Remap-SSD: Safely and Efficiently Exploiting SSD Address Remapping to Eliminate Duplicate Writes. In *Conference on File and Storage Technologies (FAST'21)*. USENIX Association, 187–202.