AutoSSD: CXL-based Autonomic and Collaborative SSD Scheduling to Reduce Tail Latency

Mingyao Shen
UC San Diego

Suyash Mahar
UC San Diego

Joseph Izraelevitz
University of Colorado, Boulder

Steven Swanson
UC San Diego

1 Introduction

With the rapid escalation in data generation [4] and the emergence of new applications leveraging vast datasets [2], the demand for efficient data storing and processing is growing correspondingly. Solid State Drives (SSDs) are a popular choice for data storage due to their superior performance. Compared to Hard Disk Drives (HDDs), SSDs offer lower access latency and higher data transfer bandwidth.

Although SSDs are highly regarded for their performance, they do not always deliver stable performance. Studies indicate that under certain conditions, the latency of an SSD can be several tens of times higher than usual, often due to internal background processes such as garbage collection and wear leveling [5]. Additionally, the issue of tail latency is more pronounced in all-SSD Redundant Arrays of Inexpensive Disks (RAID), commonly used for enhancing reliability and availability. In RAID configurations, since a single request may involve multiple SSDs, the overall latency of the request is determined by the slowest SSD. Thus, any spike in an individual SSD's latency can significantly impact the entire request's performance.

To alleviate the adverse impacts of slow SSDs in an all-SSD RAID, a straightforward approach involves redirecting requests from busy SSDs to those that are idle. Given the low likelihood of all SSDs simultaneously experiencing latency spikes, rerouting requests from SSDs with abnormally long latencies to those operating normally can ensure that requests are processed within the required time frame. However, implementing this mechanism presents several challenges:

Replication Overhead Redirecting a write request is relatively straightforward: the system simply identifies an idle SSD and forwards the request to it. However, redirecting a read request is more complex. Any SSD tasked with handling a read request must possess or be capable of reconstructing the necessary data. Consequently, the SSD designated to receive redirected read requests must maintain some form of data replication from the original data. One approach is to write a duplicate of each data block on a backup SSD. While this method is effective, it results in half of the storage space being allocated for replication, leading to a 100% increase in space overhead.

Block Tracking While redirecting write requests is straightforward, managing future read requests for the redirected data blocks introduces the need for tracking their new locations. Creating a global map that associates each block ID with an SSD block can address this issue. However, this solution faces two significant challenges. Firstly, the map would require substantial DRAM space, potentially occupying hundreds of gigabytes. This is because in a system with several tens of SSDs, there could be tens or even hundreds of billions of blocks to track. Secondly, the map could impair the performance of standard operations. Since SSDs permit concurrent request submissions from multiple CPU cores, the global map must also support concurrent updates. As map access is in the critical path of each request's processing, employing either a locking mechanism or developing a lock-free concurrent map could adversely affect the latency of all ordinary operations.

CPU Centric In current systems, CPUs exclusively handle all request scheduling tasks. This CPU-centric approach to scheduling is problematic for three reasons. Firstly, scheduling activities, such as monitoring and redirecting, can disrupt the CPUs' processing of regular tasks. Secondly, as SSDs' internal states are not transparent to CPUs, redirection happens too late for effective intervention: by the time a CPU detects an SSD's high latency and decides to reroute tasks to alternate SSDs, some requests have already been adversely affected by the overburdened SSD. Additionally, there is a delay before the CPU resumes sending requests to an SSD that was previously busy, leading to periods of underutilization. Finally, when new SSDs are added to the system, the computation power of CPUs cannot scale with the number of SSDs. This results in an increased scheduling burden on CPUs.

Redirection Performance When an SSD becomes busy, there are often already pending requests in the submission queue, leading to delays. In such cases, efficient redirection is crucial. If the CPU handles the redirection, it cannot remove requests from the submission queue, and thus, can only send duplicate requests to a backup SSD, which inefficiently consumes bandwidth. Alternatively, if the SSD itself forwards the request, it must utilize peer-to-peer direct memory ac-

cess (DMA). Each DMA operation requires initialization and setup, and it relies on interrupts to notify other devices, which introduces additional overhead.

To address these challenges, we proposed AutoSSD, a CXL-based, autonomic and collaborative SSD-centric scheduling system to reduce all-SSD RAID's tail latency. It has the following features:

CXL-based, Autonomic and Collaborative SSD-Centric Scheduling The forthcoming Compute Express Link (CXL) 3.0 protocol [1] facilitates memory sharing and coherent access between SSDs and CPUs. This enables AutoSSD to utilize shared data structures for effective CPU-SSD communication and collaboration. A lock-free queue serves as the submission queue, allowing CPUs to enqueue requests without notifying SSDs, while SSDs can continuously poll and process these requests. When an SSD becomes busy, it can cease polling from the queue and set a flag to signal CPUs and other SSDs to halt further enqueuing and redirect requests to backup SSDs where feasible. Concurrently, it can reroute existing requests in the queue to backup SSDs' queues, provided these backups are still in normal status. This SSD-centric scheduling approach, empowered by CXL, enables CPUs and SSDs to coordinate workload scheduling based on the real-time internal states of SSDs, thereby avoiding interference with regular operations and offering computational power that can scale with the number of SSDs. Moreover, the high-performance, lock-free data structures enabled by CXL's cache coherence significantly expedite redirection operations.

Using RAID Rebuild for Read Redirection AutoSSD aims to minimize the tail latency in RAID systems, which inherently possess data replication mechanisms for reconstruction. Our focus is on RAID-5 configurations. AutoSSD leverages existing parity blocks for read request redirection, eliminating the need for additional replication. In scenarios where the original SSD, designated to handle a read request, is busy, the SSD issues supplementary read requests to all SSDs holding the necessary data blocks for data reconstruction and then rebuild the data. This strategy effectively reduces space requirements for data replication and avoids imposing extra overhead on write operations.

Static + Dynamic Block Mapping AutoSSD implements a hybrid static and dynamic block mapping strategy to effectively address the block tracking challenge. Upon system initialization, a method for calculating each block's location on SSDs is established. For each RAID stripe, N locations (where N is less than or equal to the number of chunks in a stripe) are predetermined on different SSDs for write request redirection, as part of the static mapping process. This eliminates the need for a global map. In routine operations, the location of a block can be effortlessly calculated. For redirected write requests, the busy SSD employs the same method to identify a backup location, which it then records dynamically in a local set. This ensures that subsequent read requests can accurately locate the data. The combined static and dynamic mapping approach significantly reduces the DRAM space required for tracking block locations and does not adversely impact the performance of standard operations.

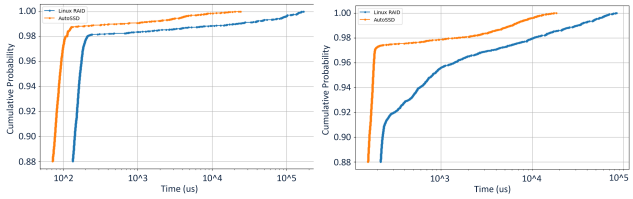

Figure 1: CDF of ioping benchmark running random write (left) and random read/write (right) on Linux RAID and AutoSSD

2 Results

To evaluate AutoSSD, we run ioping [3] on Linux mdraid and AutoSSD withour emulated system and measured their performance as shown in Figure 1. Across the workloads, we see that AutoSSD significantly outperforms Linux mdraid.

3 Conclusion

In this extended abstract, we introduce AutoSSD, a CXL-based, autonomic, and collaborative SSD-centric scheduling system designed to reduce the tail latency in all-SSD RAID configurations. AutoSSD effectively addresses key challenges associated with redirecting requests to backup SSDs, such as replication overhead, block tracking, CPU-centric issues, and redirection performance. Overall, AutoSSD demonstrates superior performance compared to traditional Linux mdraid.

References

[1] Compute express link™: The breakthrough cpu-to-device interconnect cxl™. https://www.computeexpresslink.org/, Accessed: 2023-12-22.

[2] Inside language models (from gpt to olympus). https://lifearchitect.ai/models/, Accessed: 2023-12-22.

[3] ioping: simple disk i/0 latency measuring tool. https://github.com/koct9i/ioping, Accessed: 2023-12-22.

[4] Volume of data/information created, captured, copied, and consumed worldwide from 2010 to 2020, with forecasts from 2021 to 2025. https://www.statista.com/statistics/871513/worldwide-data-created/, Accessed: 2023-12-22.

[5] Mingzhe Hao, Gokul Soundararajan, Deepak Kenchammana-Hosekote, Andrew A Chien, and Haryadi S Gunawi. The tail at store: A revelation from millions of hours of disk and {SSD} deployments. In *14th USENIX Conference on File and Storage Technologies (FAST 16)*, pages 263–276, 2016.