Fundamental OS Design Considerations for CXL-based Hybrid SSDs

Daniel Habicht*
Karlsruhe Institute of Technology

Yussuf Khalil*
Karlsruhe Institute of Technology

Lukas Werling*
Karlsruhe Institute of Technology

Thorsten Gröninger*
Karlsruhe Institute of Technology

Frank Bellosa
Karlsruhe Institute of Technology

Abstract

The first commercial implementations of CXL-based hybrid SSDs (i.e., SSDs that are both byte- and block-addressable) are looming on the horizon. Although previous works have conducted design studies on hardware concepts as well as potential use cases, none have analyzed operating system considerations and abstractions for such storage devices. We find existing abstractions (i.e., DAX in Windows and Linux) to be insufficient for hybrid SSDs and propose more appropriate resource management techniques and interfaces. In our evaluation we improve throughput by up to 4.1× for applications with strong persistence requirements using the in-memory key-value store Valkey.

1 Introduction

Hybrid SSDs are an emerging storage technology that combines *Direct Access* (DAX) known from persistent memory technologies like Intel's *Optane DCPMM* [6], with conventional block-based I/O. To enable load/store semantics on top of conventional Flash memory, hybrid SSDs feature a small on-device cache that guarantees persistency of writes.

First introduced by Bae et al. in 2018 [4], hybrid SSDs did not gain significant traction due to limitations of PCIe [8]. With the rise of the cache-coherent *Compute Express Link* (CXL) [5], this situation is about to change and first hybrid SSD offerings, like Samsung's *CXL Memory Module – Hybrid* (CMM-H) [12] or Wolley's *NVMe over CXL* (NVMe-oC) [13], are on the horizon. Contrary to PCIe, `CXL.mem` enables host-side caching of device-attached memory. Further, CXL 2.0 introduces the notion of persistent memory together with a flush-on-fail mechanism known as *Global Persistent Flush* (GPF) [5] that enables persistent CPU caches, similar to Intel's *eADR* feature on past *Optane* platforms [7].

Although several studies previously explored hybrid SSD designs as well as potential use cases [3, 4, 8, 16], they focused on hardware design aspects without exploring the design space for storage abstractions in operating systems.

In this work, we review existing OS abstractions for byte-addressable storage and explain why they are inadequate for hybrid SSDs. Based on these insights, we propose modifications to existing *POSIX* APIs for interacting with storage and present a new take on resource management of hybrid SSDs that leverages the operating system's page cache. In order to showcase the potential of hybrid SSDs for real-world workloads with strong persistence requirements, we modify the key-value store *Valkey* [1] to utilize the modified DAX interface for its *append-only file* (AOF) [2]. Our measurements show a throughput improvement of up to 4.1× for write-only workloads while also reducing the per-request CPU and energy overhead by up to 78 % and 74 %, respectively.

2 Problem Analysis

Motivated by the launch of Intel's *Optane DCPMM* [6], several operating systems, including Linux [9] and Windows [11], introduced interfaces for leveraging DAX capabilities of byte-addressable storage devices. DAX allows applications to map storage contents directly to user space bypassing the volatile OS page cache. While these abstractions work well for uniform persistent memory solutions like *Optane*, we claim that they are not suitable for hybrid SSDs because they assume a different device model.

Linux's existing DAX support, for example, assumes non-blocking load/store access on the entire storage capacity at all times [9]. Hybrid SSDs, however, can only provide this access through the much smaller cache. Consequently, when applications access an uncached file range, i.e., one that is not present in the on-device cache, through memory-mapped I/O, the resulting memory access stalls the CPU. There is no mechanism in place to defer this access to schedule another process in the meantime.

A second problem is the lack of fine-granular control over DAX. In Linux, a per-file flag determines whether storage contents are buffered in system memory or directly accessed with load/store semantics [9]. On Windows, DAX control is even more limited as DAX can only be toggled on entire NTFS volumes [14]. Assuming that the entire storage device is byte-addressable at all times, supporting more fine-granular control over DAX even when only a small subset of a file requires strong persistence guarantees of DAX provides a negligible benefit. If we apply the same approach to hybrid SSDs, pressure on the already small on-device cache is increased, leading to less effective cache usage and potentially thrashing.

*PhD student

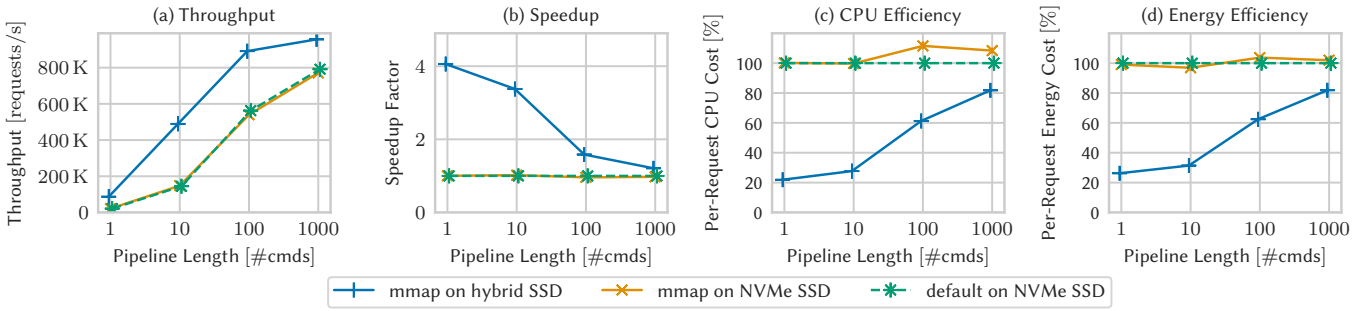

Figure 1. Performance metrics (as a function of the pipeline length) for write-only *Valkey* workload using AOF persistence.

3 Design and Implementation

Based on our problem analysis, we now propose our OS-centric approach to managing the hybrid SSD's cache as well as a revised user space interface for DAX that enables fine-granular cache control. We implement our approach for Linux version 6.6.0 [15] and the ext4 file system.

As Linux's per-file DAX flag forces applications to put more pressure on the small on-device cache than necessary, we introduce the `MAP_DAX` `mmap()` flag for requesting DAX mappings on sub-ranges of a file. This flag guarantees direct access to storage only on the requested range. In addition to the `MAP_DAX` flag, we also introduce a new resource limit (*rlimit*) for pinning DAX pages into the on-device cache using `mlock()`. As all DAX users have to share the limited cache capacity available, pinning pages to the cache can provide better performance isolation between tasks.

For supporting our proposed DAX interface in Linux, we introduce a *persistence-aware page cache*. DAX mappings established with the `MAP_DAX` flag are tracked in a per-file interval tree [10] of all DAX virtual memory areas (VMAs). The page cache uses this interval tree to determine the page type, i.e., either on-device cache for DAX mappings or volatile system memory otherwise, when allocating page cache pages. When a newly-established DAX mapping overlaps a file range buffered in volatile system memory, the DAX mapping takes precedence and forces a migration (*DAX upgrade*) of file contents from system memory to the on-device cache.

Further, the page cache tracks the mappings between cache and storage and enables the operating system to reflect the cache state in its page tables. When a load/store access misses the on-device cache, a page fault is raised and the operating system fetches the storage contents while scheduling another process, thus preventing slow synchronous memory accesses.

Apart from direct access to storage, our persistence-aware page cache can leverage persistence guarantees of the on-device cache to optimize synchronous writeback, e.g., `fsync`. During synchronous writeback, a hybrid SSD-aware file system can skip the writeback of dirty storage pages overlapped by DAX mappings. The skipped DAX pages remain dirty without breaking any of the persistence guarantees of `fsync()`. Only during asynchronous writeback, like the periodic writeback of dirty pages, or when being evicted from the page cache, file contents in the on-device cache are synced with the backing storage. With this approach, our design aims to defer the heavy cost of `fsync()` on DAX mappings to a non-performance-critical context.

4 Evaluation

We evaluate our design using the key-value store *Valkey* [1] with AOF persistence [2] and the strongest persistence configuration, i.e., calling `fsync()` after each write to the AOF. At the time of writing, there is no hybrid SSD available for purchase. Because of this, we emulate a hybrid SSD using CXL-attached DRAM for the on-device cache and a commodity NVMe SSD for the backing storage.

As *Valkey* does not use memory-mapped I/O for writing to the AOF, we implement an AOF mmap backend that can optionally use the `MAP_DAX` flag for the AOF mapping. Our AOF mmap backend only keeps the most recent AOF contents in the page cache (at most 40 MiB) and resizes the AOF in a background thread when running out of space.

Figure 1 shows our measurements of the throughput as well as the per-request CPU and energy overhead for the mmap backend and the default AOF implementation on a write-only workload. Our design improves the throughput by up to 4.1× while reducing the CPU and energy overhead by up to 78 % and 74 %, respectively. As the number of AOF writes per request decreases with the pipeline length, and with it the `fsync()` overhead, the impact of the hybrid SSD decreases.

5 Conclusion

In this work, we explored emerging hybrid SSDs from the operating system's perspective. Based on our analysis of existing operating systems, we identified insufficient resource management in existing DAX abstractions as a major obstacle in the adoption of hybrid SSDs and proposed an OS-centric design for managing hybrid SSDs as well as a revised user space interface for DAX. Our evaluation using *Valkey* with an emulated hybrid SSD shows that our design can improve throughput by up to 4.1× on workloads with strong persistence requirements.

References

[1] 2024. *Valkey: an open source, in-memory data store.* https://valkey.io/

[2] 2024. *Valkey Persistence.* https://valkey.io/docs/topics/persistence/

[3] Ahmed Abulila, Vikram Sharma Mailthody, Zaid Qureshi, Jian Huang, Nam Sung Kim, Jinjun Xiong, and Wen-mei Hwu. 2019. FlatFlash: Exploiting the Byte-Accessibility of SSDs within a Unified Memory-Storage Hierarchy. In *Proceedings of the Twenty-Fourth International Conference on Architectural Support for Programming Languages and Operating Systems* (Providence, RI, USA) *(ASPLOS '19)*. Association for Computing Machinery, New York, NY, USA, 971–985. https://doi.org/10.1145/3297858.3304061

[4] Duck-Ho Bae, Insoon Jo, Youra Adel Choi, Joo-Young Hwang, Sangyeun Cho, Dong-Gi Lee, and Jaeheon Jeong. 2018. 2B-SSD: The Case for Dual, Byte- and Block-Addressable Solid-State Drives. In *2018 ACM/IEEE 45th Annual International Symposium on Computer Architecture (ISCA)*. IEEE, Los Angeles, CA, 425–438. https://doi.org/10.1109/ISCA.2018.00043

[5] CXL Consortium. 2023. *Compute Express Link Specification Revision 3.1.*

[6] Intel Corporation. 2019. *Intel® Optane™ DC Persistent Memory Product Brief.* https://www.intel.com/content/dam/www/public/us/en/documents/product-briefs/optane-dc-persistent-memory-brief.pdf

[7] Intel Corporation. 2021. *eADR: New Opportunities for Persistent Memory Applications.* https://www.intel.com/content/www/us/en/developer/articles/technical/eadr-new-opportunities-for-persistent-memory-applications.html

[8] Myoungsoo Jung. 2022. Hello bytes, bye blocks: PCIe storage meets compute express link for memory expansion (CXL-SSD). In *Proceedings of the 14th ACM Workshop on Hot Topics in Storage and File Systems* (Virtual Event) *(HotStorage '22)*. Association for Computing Machinery, New York, NY, USA, 45–51. https://doi.org/10.1145/3538643.3539745

[9] Linux kernel contributors. 2023. *Direct Access for Files – The Linux Kernel Documentation.* https://docs.kernel.org/6.6/filesystems/dax.html

[10] Rob Landley. 2023. *Red-Black Trees (Rbtree) in Linux – The Linux Kernel Documentation.* https://docs.kernel.org/6.6/core-api/rbtree.html

[11] Microsoft Corporation. 2022. *Understand Direct Access (DAX) and create DAX volumes with persistent memory devices.* https://learn.microsoft.com/en-us/windows-server/storage/storage-spaces/persistent-memory-direct-access

[12] Rekha Pitchumani. 2023. *CMM-H (CXL Memory Module – Hybrid): Samsung's CXL-based SSD for the Memory-centric Computing Era.* Samsung. https://semiconductor.samsung.com/us/news-events/tech-blog/webinar-memory-semantic-ssd/

[13] Bernard Shung, San Chang, and Terry Cheng. 2023. *NVMe over CXL (NVMe-oC): An Ultimate Optimization of Host-Device Data Movement.* https://sc23.supercomputing.org/proceedings/exhibitor_forum/exhibitor_forum_files/exforum118s2-file2.pdf

[14] Tom Talpey. 2017. Persistent Memory in Windows Server 2016. In *Persistent Memory Summit 2017*. SNIA.org, SNIA 5201 Great America Parkway Suite 320 Santa Clara, CA 95054, 23 pages. https://www.snia.org/sites/default/files/PM-Summit/2017/presentations/Tom_Talpey_Persistent_Memory_in_Windows_Server_2016.pdf

[15] Linus Torvalds. 2023. *Linux Kernel.* https://git.kernel.org/pub/scm/linux/kernel/git/stable/linux.git/tree/?h=v6.6

[16] Shao-Peng Yang, Minjae Kim, Sanghyun Nam, Juhyung Park, Jin yong Choi, Eyee Hyun Nam, Eunji Lee, Sungjin Lee, and Bryan S. Kim. 2023. Overcoming the Memory Wall with CXL-Enabled SSDs. In *2023 USENIX Annual Technical Conference (USENIX ATC 23)*. USENIX Association, Boston, MA, 601–617. https://www.usenix.org/conference/atc23/presentation/yang-shao-peng