When Caching Systems Meet Emerging Storage Devices: A Case Study

Zhen Lin[†*], Lianjie Cao[†], Faraz Ahmed[†], Hui Lu[*], Puneet Sharma[†]
[†]Hewlett Packard Labs, [*]Binghamton University

ABSTRACT

Block-layer caching systems improve the I/O performance by using hybrid storage devices; the advent of fast, byte-addressable storage enables caching systems to further leverage new storage tiers (*e.g.*, with persistent memory as the cache device and SSD as the backend device) to achieve better caching performance. However, the new storage devices also challenge the design and implementation of existing block-based caching systems. This paper conducts a comprehensive performance study of a popular caching system, Open CAS, and identifies new, unrevealed software bottlenecks. Our observations and root cause analysis cast light on optimizing the software stack of caching systems to incorporate emerging storage technologies.

CCS CONCEPTS

• **Information systems → Hierarchical storage management**; **Storage architectures**.

KEYWORDS

Caching, Persistent Memory, Open CAS

1 INTRODUCTION

Caching systems are implemented at various layers of the storage stack to improve the performance of storage systems. For example, the Linux operating system uses DRAM as a volatile cache (*i.e.*, page cache), facilitating faster I/O than directly accessing slower, bulky storage devices (*e.g.*, HDDs). However, DRAM is expensive and has limited capacity. Further, as fast Solid-State Drives (SSDs) are widely available, caching implemented at the block level becomes attractive – SSDs are much faster than HDDs, provide higher capacities, and cost less per GB of storage compared to DRAM. However, SSDs are still more expensive than HDDs where large storage capacities are needed.

To achieve high I/O performance while reducing monetary costs, block-layer caching systems (*e.g.*, bcache[1], dm-cache[4], and dm-writecache[5]) enable the use of hybrid storage, where a fast persistent block device can be used for caching and a slower one used for large storage capacity. For example, Open CAS [11], an advanced caching solution recently developed by Intel, allows different cache modes and the use of various caching policies based on I/O workload characteristics for better performance.

While the majority of existing caching systems are primarily designed to improve the performance of HDDs (*i.e.*, backend devices) by caching data on SSDs (*i.e.*, cache devices), the advent of *fast* storage and interconnect technologies, such as persistent memory (PMem) [7], Non-Volatile Memory Express (NVMe) [6], and Compute Express Link (CXL) [3], present new opportunities but also pose critical challenges. On the one hand, caching systems can leverage new tiered storage devices for cache devices (*e.g.*, PMem) and backend devices (*e.g.*, SSDs), accelerating caching performance. On the other hand, the overhead of the software stack and the block-centered design of caching systems may prevent it from fully unlocking the capabilities of fast and/or byte-addressable storage devices.

In this paper, we use Open CAS as an example caching system and perform a thorough performance study using Intel Optane PMem as the cache device and various types of SSDs as the backend device. Our study unveils important, previously undetected software bottlenecks in Open CAS: 1) In addition to data updates, the *block-based* metadata updates (*i.e.*, for maintaining the information of cached data) in Open CAS consume significant cache device bandwidth – *e.g.*, same as data updates. 2) The cache eviction operation in Open CAS further deteriorates this situation by involving even more metadata updates (*e.g.*, 2×). 3) Caching systems that incorporate optimizations with outdated assumptions make it challenging to predict performance accurately. 4)

HotStorage '23, July 9, 2023, Boston, MA, USA

ACM ISBN 979-8-4007-0224-2/23/07…$15.00
https://doi.org/10.1145/3599691.3603413

Open CAS lacks effective support for both emerging storage technologies and changing workloads.

We believe our observations and result analysis will cast light on optimizing caching systems to well support both emerging hardware and dynamic workloads.

2 BACKGROUND AND RELATED WORK

Fast storage and interconnect: With new storage technologies like 3D XPoint [10], NVMe SSDs (over the PCIe bus) achieve much higher bandwidth (*e.g.*, 8 GB/s) and lower latency (*e.g.*, 3μs) [25]. Further, byte-addressable persistent memory (PMem) has also been commercially available [7], allowing programs to access data in non-volatile memory directly from CPU using `load` and `store` instructions. PMem offers an order of magnitude higher capacity than DRAM and within an order of magnitude performance of DRAM [23].

Though Intel discontinues its Optane product recently, the storage community is moving to investigate high-speed CPU-to-device interconnect [8], *i.e.*, Compute Express Link (CXL) [3]. CXL provides a more unified interface to disaggregate various types of storage devices, such as DRAM, PM, and PCIe devices, directly to CPU. CXL also has the potential to replace PCIe storage's block interface with a memory-like, byte-addressable interface with minor modifications [18]. We envision that *storage devices will continue to evolve, becoming increasingly faster, offering higher bandwidth, lower latency, and providing byte-addressability through a memory-like interface*. While our study primarily centers around PMem (connected via the memory bus), we believe it provides valuable insights applicable to the utilization of future fast, byte-addressable storage devices, such as those leveraging Compute Express Link (CXL).

Block-level caching systems: Storage-aware caching systems comprise multiple tiers of storage devices, including DRAM, PMem, SSD, HDD, etc. The faster devices are more expensive and used at higher tiers than slower devices. In the context of high-performance devices in multi-tier caching, there have been work targeting different layers of the storage stack, *e.g.*, Open CAS [11], OrthusCAS [22], and NVCache [15]. They are cache acceleration frameworks implemented at the block level that uses a high-performance block device on the local host as a cache for a slower block device. OrthusCAS [22], when under heavy load, can offload excessive I/Os from the caching layer to the capacity layer. It can be integrated with Open CAS with minor modifications. Hence most of our observations should also apply to OrthusCAS. MapperX is an extension to the Linux kernel's DM-cache component and improves the crash recovery time by addressing the drawbacks of asynchronous metadata updates in DM-cache [24]. First Responder is a caching framework implemented at the Virtual File System (VFS) layer that uses fast PMem devices as a buffer for traditional file systems like EXT4 [21]. Strata is a local file system that works with multi-tier storage, including PMem; it implements a file system that spans both user and kernel space with caching [20]. Assise is a distributed file system that uses PMem as a fast and persistent client-local caching device [14].

Open CAS: Open Cache Acceleration Software, short for Open CAS [11], is a block-level caching solution primarily designed to improve the performance of HDDs (backend devices) by caching data on SSDs (cache devices). Open CAS is widely used and studied in both industrial solutions [12, 13] and research projects [16, 22]. There are two different implementations of Open CAS – SPDK and Open CAS Linux. Our study focuses on Open CAS Linux. Open CAS Linux is a kernel module that works under Linux in-kernel file systems (*e.g.*, EXT4, XFS, and Btrfs). It handles I/O requests from upper-layer file systems through a CAS virtual block device and submits I/O requests to underlying storage devices according to cache modes and customized cache rules. Open CAS supports six cache modes, and this paper, due to the page limit, focuses on the most commonly used cache mode – Write-Back (WB).

3 MEASUREMENT STUDY

This section presents a comprehensive performance study and analysis of Open CAS with Intel Optane Persistent Memory as the cache device. We use it as a case study to understand the performance of emerging storage devices on modern caching systems in various conditions and configurations. Based on the performance results, we provide insights to guide the redesign or optimization of existing caching systems to release the full power of emerging storage devices. This performance study focuses on write performance evaluation because writes are more complicated than reads and often cause system performance bottlenecks.

3.1 Experimental Configurations

The experiments are conducted on an HPE ProLiant DL380 Gen10 Plus server with 1 × Intel Xeon Silver 4314 processor, 4 × 32GB DDR4 3200MHz DRAM, and 4 × 256GB Intel Optane 200 Persistent Memory. For comparison, the server also includes 1 × 400GB Intel Optane NVMe SSD P5800X, 1 × 960GB Samsung NVMe SSD PM9A3, and 1 × 960GB Samsung SATA SSD PM863. Ubuntu Server 20.04.6 LTS is installed with 5.4.0-144-generic Linux Kernel and Ext4 as the file system (with journaling disabled). To generate write workload, we use FIO (version 3.34-13) in single thread mode with 1) direct I/O (`-direct=1`) to bypass page cache and 2) synchronous data I/O (`-sync=dsync`). Since Open CAS is designed to support SSDs and HDDs, Optane PMem can only

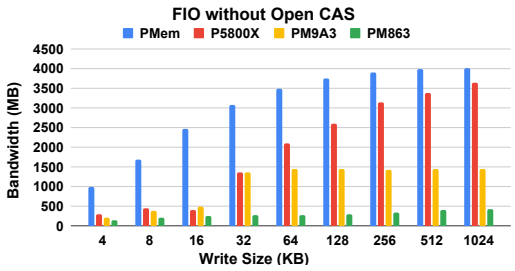


Figure 1: FIO experiments without Open CAS.

be used as a regular block device – direct access (DAX) mode is not supported.

Before running the tests, we pre-create an 8GB test file and flush it to the backend device (*i.e.*, thus to avoid filesystem metadata updates during the write operations, as we mainly focus on data updates). The experiments are carefully crafted to cover different scenarios. The capacity of the cache device is set to 4GB and FIO is used to write a file with different sizes (from 2GB to 6GB) to the Open CAS virtual block device. To further understand the impacts of various factors, we also test different cache line sizes (CL = 4KB, 16KB, and 64KB), various single write sizes (WS from 4KB to 1024KB), and multiple accesses of the same file.

3.2 Raw Performance of Storage Devices

We first benchmark the write bandwidth of all storage devices *without* Open CAS using FIO to generate write requests of different sizes. The results are used as the baseline when compared to Open CAS. As illustrated in Figure 1, PMem significantly outperforms all types of SSDs, especially with small write sizes. Compared to P5800X which uses the same 3D XPoint technology, PMem still consistently exhibits better write bandwidth across all write sizes. This is mainly because although they both use Intel Optane memory media, the Optane PMem is in a DIMM package and operates on the DRAM bus (*i.e.*, NVDIMM) while Optane P5800X SSD resides in a standard NAND package model (*i.e.*, U.2) on the PCIe bus using the NVMe protocol [9]. The results are also consistent with previous studies [17]. The results indicate that PMem is an ideal option for cache devices due to its high performance.

3.3 Open CAS Performance

Our performance study of Open CAS focuses on the Write-Back (WB) mode. The WB engine, the major component of WB mode, first writes data to the cache device in one or multiple cache lines and acknowledges the application before the data is written to the backend device. A cache line is the smallest data unit that Open CAS can manage. Every occupied cache line is associated with a core line on

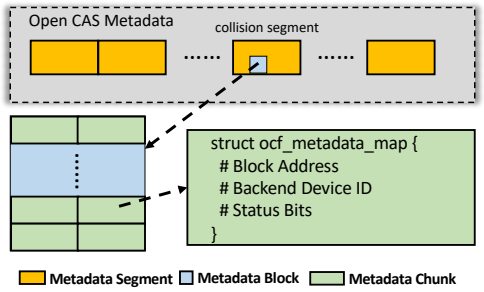


Figure 2: Open CAS Metadata Structure

the backend device and the cache line will be eventually written back to its associated core line. Open CAS supports different cache line sizes ranging from 4KB to 64KB. Note that the cache line size cannot be changed once the virtual block device of Open CAS is initialized. The dirty data on the cache device are flushed to the backend device periodically by the cleaner in the background [1].

If data is written to an occupied cache line, it is a cache hit. Otherwise, it is a cache miss. If the cache device is full, cache evictions are triggered to clear some occupied cache lines based on the default LRU eviction policy. To measure the performance of Open CAS with more sophisticated cache hit and cache eviction scenarios, we consecutively write the same file twice *i.e.*, the 1st access and 2nd access, combined with different cache line sizes (CL = 4KB to 64KB) and different file sizes (FS = 2GB to 6GB while the capacity of cache device (CC) is 4GB). We will particularly examine the following four cases:

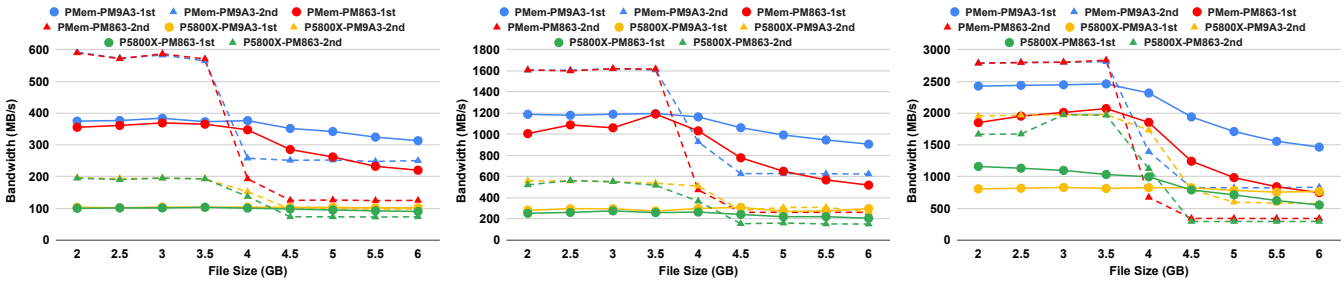
- 1st access and $FS < CC$, full cache miss and no cache eviction.
- 1st access and $FS \geq CC$, full cache miss and cache evictions for the last part of the file.
- 2nd access and $FS < CC$, full cache hit and no cache eviction.
- 2nd access and $FS \geq CC$, full cache miss and possibly cache evictions for the entire file.

3.3.1 *Metadata Overhead.*

Observation: Figure 3 shows the write bandwidth when $WS = CL$, meaning each write request accesses exactly one cache line. When $FS < CC$, the 1st access yields lower bandwidth than the 2nd access in all cases, but the gap between the two accesses shrinks as CL and WS increase from 4KB to 64KB. For example, with PMem+PM9A3, the bandwidth gap changes from 40%@4KB, 25%@16KB, to 14%@64KB.

Analysis: For the 1st access, due to cache misses, the metadata of Open CAS needs to be updated and flushed to the

[1]To ensure great control over the study of Open CAS behavior, the cleaner process is disabled in all the experiments.



(a) CL = WS = 4KB (b) CL = WS = 16KB (c) CL = WS = 64KB

Figure 3: Write bandwidth of Open CAS Write-Back mode when WS = CL

cache device for each data write. For the 2nd access, no metadata operation is involved due to a full cache hit.

More specifically, as shown in figure 2, Open CAS has different types of metadata information stored in multiple segments, among which the "collision" segment has the most profound performance impact based on our analysis. The collision section includes a set of 4KB metadata blocks (in DRAM), each of which consists of multiple 12-byte chunks (12, 18, and 42 bytes for CL = 4KB, 16KB, and 64KB respectively) storing the mapping between a cache_line and a core_line and status bits (valid and dirty).

Each write request in the 1st access needs to 1) update a collision block of 12 bytes, 2) overwrite the 4KB metadata block where the collision metadata chunk resides, and 3) flush the corresponding 4KB metadata block to the cache device. Therefore, the total number of bytes written to the cache device for a single write in the 1st access is 4KB metadata + cache-line-size data. In the case of CL = 4KB, it is $2 \times FS$ written to the cache device for the 1st access. For example, when $CL = WS$ = 4KB, the difference of write bandwidth between 1st and 2nd access for PMem and P5800X are 40% and 48%. The impact of the extra metadata write in the 1st access is amortized as CL increases, which explains why the difference shrinks as we mentioned above.

Table 1: CPU usage breakdown

	Raw1	WB_1st	WB_2nd
FIO	15%	7%	11%
OS	3%	3%	7%
VFS	3%	2%	3%
ext4	16%	11%	16%
Open CAS	N/A	45%	38%
Block I/O	54%	25%	20%
PMem	9%	7%	5%

Table 1 shows the CPU usage breakdown (via profile [2]) for the FIO raw performance, 1st, and 2nd accesses when $FS < CC$. We observe that Open CAS takes up significant CPU time in both 1st (45%) and 2nd (38%) accesses (including both metadata and data operations). While it is hard to get the portion caused by the metadata operations accurately, our estimation shows that metadata takes up to 50% of the CPU usage consumed by Open CAS in the 1st access.

Takeaway #1: *The metadata updates of Open CAS use additional cache device bandwidth. Meanwhile, the software stack of Open CAS consumes a significant amount of CPU resources.*

3.3.2 *Cache Eviction Overhead.*

Observation: Figure 3 also shows that when $FS \geq CC$, the bandwidth of the 1st access drops gradually as file size increases, while the bandwidth of the 2nd access drops drastically (compared to when $FS < CC$) and keeps around the same value as file size increases. The same pattern applies to all the combinations of cache and backend devices in our experiments. But switching the backend device from PM9A3 (NVMe SSD) to PM863 (SATA SSD) brings a noticeable difference that increases as CL and WS increase.

Analysis: The major difference when $FS > CC$ is that cache evictions are triggered. For the 1st access, based on the default LRU eviction policy, the oldest cache lines need to be evicted for writing the last part of the file. While for the 2nd access, all data written to the cache device during the 1st access need to be evicted, leading to cache eviction for every write request. For example, with FS = 5GB, the first 1GB data need to be evicted and replaced with the last 1GB data in the 1st access. When the 2nd access starts, there is no cache hit for the 1st GB of the file because it is already evicted at the end of the 1st access. Hence, the 2nd GB of the file needs to be evicted based on LRU. This process continues throughout the entire 2nd access. Hence, the different write bandwidth values of the 1st and 2nd accesses are mostly due to distinct cache eviction behaviors. A cache eviction in Open CAS involves multiple steps:

(1) Read old data from cache device to DRAM (CL size).
(2) Write old data to the backend device (CL size).

(3) Update status bits of old metadata and write it to cache device (4KB).
(4) Write new data to cache device (CL size).
(5) Write new metadata to cache device (4KB).

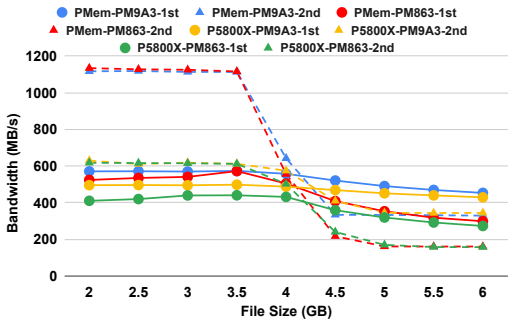
In total, there is CL read from the cache device + ($CL + 4KB + 4KB$) write to the cache device + CL write to the backend device. Notice again, *to update the valid bit, it is not the 12-byte collision metadata chunk that needs to be updated and flushed to the cache device, but the entire 4KB collision metadata block.* In the worst case (*i.e.*, $WS = CL = 4$KB), the total number of bytes written to the cache device can be 3× for a write request with cache eviction. If WS or $CL > 4$KB, this overhead is amortized due to larger data size or because the collision metadata block of multiple cache lines for the same write request may share the same 4KB page ($WS > CL$).

To summarize, the write bandwidth, when $FS \geq CC$, mainly depends on three factors: 1) the write bandwidth of the first 4GB of the file, 2) the overhead of the write amplification to the cache device caused by cache eviction (up to 3 times), and 3) the write bandwidth of the backend device to handle the old data flush caused by cache eviction. For the 1st access, cache eviction happens when writing beyond the cache capacity. Hence the impact of cache eviction increases with larger file sizes, leading to a gradually decreasing write bandwidth. While for the 2nd access, every write request triggers a cache eviction. Hence, the impact of cache eviction stays roughly the same throughout the 2nd access, leading to a consistently low write bandwidth.

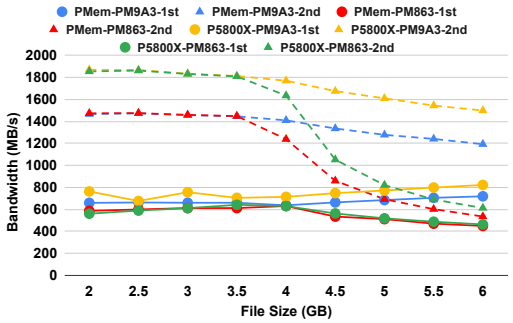
Takeaway #2: *Cache eviction triggers even more metadata updates, resulting in significant write amplification and overhead.*

3.3.3 *Write Invalidate.*

Observation: Unlike the experiments shown in Figure 3 where $WS = CL$, the real-world workloads often consist of different write sizes other than the cache line size. We observe that when one write request requires multiple cache lines (*i.e.*, $WS > CL$), the behavior and performance of Open CAS change significantly. Figure 4 shows the write bandwidth of Open CAS when one write request needs 4 and 128 cache lines, respectively. As we change the write size from $WS = CL$ (Figure 3a) to $WS = 4 \times CL$ (Figure 4a), a similar pattern is observed (Section 3.3.2). However, when the write size is changed to $WS = 128 \times CL$ (Figure 4b), the same pattern does not fully apply. When $FS < CC$, the write bandwidth of the 2nd access remains better than the 1st access as the 1st access involves higher overhead due to metadata updates as we discussed (Section 3.3.2). However, when $FS \geq CC$, the write bandwidth of the 1st access no longer drops gradually. In fact, it slightly increases when PM9A3 is used as the backend device. In contrast, for the 2nd access, the write bandwidth starts to drop gradually as file size increases. The other observation is when $WS = 512$KB, P5800X outperforms PMem in the 2nd access by ~25%.



(a) CL = 4KB and WS = 16KB



(b) CL = 4KB and WS = 512KB

Figure 4: Write bandwidth of Open CAS Write-Back mode when WS > CL.

Analysis: After analyzing Open CAS statistics and source code, we found it is mainly caused by the Write-Invalidate (WI) mode of Open CAS and the performance characteristics of the storage devices. WI mode can be triggered in several different ways and it passes write requests directly to the backend device. In our experiments, WI mode is triggered if 32 cache line evictions cannot provide enough free cache lines to accommodate the write request (32 is defined by a constant variable). More specifically, when a write request of 512KB is received, the WB engine tries to evict up to 32 cache lines based on LRU. Since the write request requires 128 4KB cache lines, WI mode is invoked and data is directly written to the backend device. WI mode can reduce the number of cache evictions and avoid the overhead discussed in Section 3.3.2. However, if the performance of the backend device is not desirable (low), it can still lead to low write bandwidth.

For PMem + PM9A3, when $FS > CC$ and $WB = 512$KB, during the 1st access the WB engine tries to evict cache lines from PMem beyond the 4th GB of the file. But since it requires more than 32 cache line evictions, WI mode is invoked and data are directly written to PM9A3 without

incurring the metadata and eviction overhead. Hence, at the end of the 1st access, the first 4GB of the file is stored on PMem, and the rest of the file is stored on PM9A3. The write bandwidth of the 1st access is slightly higher than when $FS < CC$ because the performance of PM9A3, @512KB, is slightly better than the PMem with the metadata overhead. For the 2nd access, it is no longer full cache eviction as we discussed in Section 3.3.2 because the last part of the file was written to PM9A3 rather than PMem in the 1st access. Hence, it is cache hit for the first 4GB of the file, and data is written to PMem without metadata overhead. The rest of the file is written to PM9A3 due to WI, similar to what happened in the 1st access. Since the write bandwidth to PMem with cache hit is better than PM9A3 in WI, the overall write bandwidth of the 2nd access decreases gradually as file size increases, *i.e.*, more data need to be written directly to PM9A3 by WI. In this experiment, the performance of the backend device is critical. When PM9A3 is replaced with PM863 with lower performance, the write bandwidths of both the 1st and 2nd drop more significantly than PM9A3 when $FS > CC$.

Takeaway #3: *Static optimizations and configurations of caching systems without considering the interplay of workload and storage device performance characteristics may yield intricate behaviors, leading to undesired performance.*

We also observe that PMem yields slightly lower write bandwidth than P5800X for the 2nd access when $WS = 512KB$, and it is probably because Open CAS uses PMem as a regular block device. In Open CAS, cache lines are divided into multiple lists. If a write request involves multiple cache lines, free cache lines are selected from those lists, and the write is grouped into a smaller number of mini-batch writes. With P5800X as the cache device, the NVMe driver handles the mini-batch writes in a non-blocking manner, while PMem has to handle multiple writes sequentially on CPU leading to lower performance. Also, since PMem is used as a block device by Open CAS, a considerable amount of CPU resources are spent on the block I/O layer which can be minimized with PMem DAX mode.

Takeaway #4: *Emerging storage devices are not explicitly well supported and optimized by caching systems leading to discounted performance behaviors.*

4 INSIGHTS AND CONCLUSIONS

In this paper, we present a detailed performance study of Open CAS on Intel Optane PMem and conclude four takeaways. Based on our analysis, we present the following insights for designing and/or optimizing caching systems.

First, as Open CAS has been originally built for block devices, it is agnostic of emerging storage devices like PMem. Simply communicating with PMem through a block-layer interface is extremely inefficient. Instead, to fully explore the performance of PMem, a caching system should enable the direct access (DAX) mode for PMem via its byte-addressability interface. Caching devices and backend devices also need to be handled differently for devices like PMem, because the I/O processing of such devices happens on-CPU using load/store while the processing for SSDs and HDDs can be partially offloaded to the storage controllers.

Second, Open CAS involves a large number of metadata updates to persist cache line information. While the update to each metadata involves a small number of bytes, the block access granularity of Open CAS results in significant write amplification. As byte-addressability is a key feature provided by PMem and future CXL-connected SSDs [19], a caching system should leverage such a byte-addressable interface for more fine-grained metadata updates.

Last, providing a "one-size-fits-all" optimization may not be possible for caching systems, given that we have more diverse and heterogeneous storage devices and dynamic workloads. More "tunable" optimization approaches by considering the storage performance characteristics and workload composition might help.

REFERENCES

[1] Bcache. https://bcache.evilpiepirate.org/.
[2] Bcc profile tool. https://github.com/iovisor/bcc/blob/master/tools/profile.py.
[3] Compute express link™: The breakthrough cpu-to-device interconnect. https://www.computeexpresslink.org/download-the-specification.
[4] Dmcache. https://www.kernel.org/doc/Documentation/device-mapper/cache.txt.
[5] Dmwritecache. https://docs.kernel.org/admin-guide/device-mapper/writecache.html.
[6] Intel optane dc ssd series. https://www.intel.com/content/www/us/en/products/details/memory-storage/data-center-ssds/optane-dc-ssd-series.html.
[7] Intel optane dimm. https://www.intel.com/content/www/us/en/architecture-and-technology/optane-dc-persistent-memory.html.
[8] Intel optane is winding down. what's that mean for you and your customers? https://www.techproviderzone.com/cloud-and-data-centers/intel-optane-is-winding-down-what-s-that-mean-for-you-your-customers.
[9] Intel optane persistent memory overview. https://www.intel.com/content/www/us/en/products/docs/memory-storage/optane-persistent-memory/overview.html.
[10] Intel® optane™ ssd p5800x series. https://www.intel.com/content/www/us/en/products/docs/memory-storage/solid-state-drives/data-center-ssds/optane-ssd-p5800x-p5801x-brief.html.
[11] Open CAS - open cache acceleration software. https://open-cas.github.io/index.html#who-already-uses-ocf.
[12] Open CAS - who already uses ocf? https://open-cas.github.io/ocf_intro.html#who-already-uses-ocf.
[13] Research on performance tuning of hdd-based ceph cluster using open cas. https://01.org/blogs/tingjie/2020/research-performance-tuning-hdd-based-ceph-cluster-using-open-cas.
[14] Anderson, T. E., Canini, M., Kim, J., Kostić, D., Kwon, Y., Peter, S., Reda, W., Schuh, H. N., and Witchel, E. Assise: Performance and availability via client-local nvm in a distributed file system. OSDI'20,

USENIX Association.

[15] Dulong, R., Pires, R., Correia, A., Schiavoni, V., Ramalhete, P., Felber, P., and Thomas, G. Nvcache: A plug-and-play nvmm-based i/o booster for legacy systems. In *2021 51st Annual IEEE/IFIP International Conference on Dependable Systems and Networks (DSN)* (2021), IEEE, pp. 186–198.

[16] Fedorova, A., Smith, K. A., Bostic, K., LoVerso, S., Cahill, M., and Gorrod, A. Writes hurt: lessons in cache design for optane nvram. In *Proceedings of the 13th Symposium on Cloud Computing* (2022), pp. 110–125.

[17] Izraelevitz, J., Yang, J., Zhang, L., Kim, J., Liu, X., Memaripour, A., Soh, Y. J., Wang, Z., Xu, Y., Dulloor, S. R., et al. Basic performance measurements of the intel optane dc persistent memory module. *arXiv preprint arXiv:1903.05714* (2019).

[18] Jung, M. Hello bytes, bye blocks: Pcie storage meets compute express link for memory expansion (cxl-ssd). HotStorage '22, Association for Computing Machinery, p. 45–51.

[19] Jung, M. 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* (2022), pp. 45–51.

[20] Kwon, Y., Fingler, H., Hunt, T., Peter, S., Witchel, E., and Anderson, T. Strata: A cross media file system. In *Proceedings of the 26th Symposium on Operating Systems Principles* (2017), pp. 460–477.

[21] Song, H., Kim, S., Kim, J. H., Park, E. J., and Noh, S. H. First responder: Persistent memory simultaneously as high performance buffer cache and storage. In *USENIX Annual Technical Conference* (2021), pp. 839–853.

[22] Wu, K., Guo, Z., Hu, G., Tu, K., Alagappan, R., Sen, R., Park, K., Arpaci-Dusseau, A. C., and Arpaci-Dusseau, R. H. The storage hierarchy is not a hierarchy: Optimizing caching on modern storage devices with orthus. In *19th USENIX Conference on File and Storage Technologies (FAST 21)* (2021), pp. 307–323.

[23] Yang, J., Kim, J., Hoseinzadeh, M., Izraelevitz, J., and Swanson, S. An empirical guide to the behavior and use of scalable persistent memory. In *18th USENIX Conference on File and Storage Technologies (FAST 20)* (Santa Clara, CA, Feb. 2020), USENIX Association, pp. 169–182.

[24] Yin, L., Wang, L., Zhang, Y., and Peng, Y. MapperX: Adaptive metadata maintenance for fast crash recovery of DM-Cache based hybrid storage devices. In *USENIX Annual Technical Conference (USENIX ATC 21)* (2021), pp. 705–713.

[25] Zhong, Y., Li, H., Wu, Y. J., Zarkadas, I., Tao, J., Mesterhazy, E., Makris, M., Yang, J., Tai, A., Stutsman, R., and Cidon, A. XRP: In-Kernel storage functions with eBPF. In *16th USENIX Symposium on Operating Systems Design and Implementation (OSDI 22)* (Carlsbad, CA, July 2022), USENIX Association, pp. 375–393.