Cheetah: An Adaptive User-Space Cache for Non-volatile Main Memory File Systems

Tian Yan[1], Linpeng Huang[1](✉), and Shengan Zheng[2]

[1] Department of Computer Science and Engineering, Shanghai Jiao Tong University, Shanghai, China
{officialyan,lphuang}@sjtu.edu.cn

[2] Department of Computer Science and Technology, Tsinghua University, Beijing, China
venero@tsinghua.edu.cn

Abstract. Over the past decade, most NVMM file systems have been designed without detailed knowledge of real NVDIMMs. With the release of Intel Optane DC Persistent Memory, researchers find that the performance characteristics of real NVMM differ a lot from their expectations. The design decisions they made lead to limited scalability, significant software overhead, and severe write amplification.

We present Cheetah, a user-level cache designed for existing NVMM file systems to improve overall performance. Cheetah leverages the unique characteristics of Intel Optane DC persistent memory to design a fine-grained data block allocation policy in order to reduce write amplification. To minimize the impact of the long write latency of NVMM, Cheetah absorbs asynchronous writes in DRAM rather than NVMM. Our experimental results show that Cheetah provides up to 3.5× throughput improvement compared to the state-of-the-art NVMM file systems in write-intensive workloads.

Keywords: Non-volatile main memory · File system · Cache scheme

1 Introduction

Emerging fast, byte-addressable non-volatile main memory (NVMM), such as PCM [6] and Intel Optane DC persistent memory (Optane DCPMM [5]), is expected to revolutionize the memory and storage hierarchy of existing computer systems. NVMM can be directly placed on the memory bus, thereby allowing applications to access data via processor `load/store` instructions. Compared with traditional disks (HDDs and SSDs), NVMM offers lower latency and higher bandwidth while providing direct access.

Over the past decade, several state-of-the-art NVMM-aware file systems, such as BPFS [7], PMFS [8], and NOVA [16], have been proposed. These file systems leverage the Direct Access (DAX) feature of NVMM to avoid unnecessary copy.

Since NVMMs are directly attached to the main memory bus, NVMM-aware file systems allow users to access file data directly without the OS page cache.

Unfortunately, existing NVMM file systems have two major drawbacks. First, researchers find that the real NVDIMMs' performance is far from their expectations, especially its concurrency and write performance. The experimental results revealed in a recent report [17] exhibit NVMM has only about one-sixth write bandwidth than DRAM. Second, for different applications with various access patterns, the conventional fixed-size block allocation scheme is inefficient. Recent studies [15,17] show that Optane DCPMM utilizes 256-byte internal blocks to access data and has several storage mechanisms different from DRAM. Uniformed block allocation in file systems is not suitable for small writes since it causes severe write amplification.

To address the drawbacks, we propose Cheetah, a user-level cache for NVMM file systems. Cheetah combines the benefits of DRAM and NVMM to improve overall performance. To hide the long write latency of NVMM, Cheetah utilizes a small DRAM cache to absorb asynchronous writes. Cheetah offers an adaptive data block allocation policy for different writes to minimize write amplification. Cheetah is compatible with most kernel NVMM-aware file systems, such as NOVA [16], EXT4-DAX [2], XFS-DAX [4], etc.

Cheetah distinguishes itself from traditional DRAM cache schemes in several aspects. First, Cheetah adopts a more accurate profiler for caching the most suitable data in DRAM. Second, Cheetah adopts an adaptive cache replacement strategy, which dynamically adjusts the impact of different access patterns on data replacement. The contributions of this paper include:

- We design a profiler that accurately profiles I/O frequency of file data.
- We design a hybrid memory management module to manage file metadata and data on NVMM and DRAM effectively.
- We implement Cheetah and evaluate it by using a collection of micro-benchmarks and real applications. We find that Cheetah exhibits higher performance than the state-of-the-art file systems on a range of workloads.

2 Background

2.1 Non-volatile Main Memory

Non-volatile main memory (NVMM) is an emerging memory technology that provides byte-addressability and durability. NVMM also allows direct data access via processor `load` and `store` instructions. However, the past researches for NVMM have been made without detailed knowledge of real NVMM. Recent works [13–15,17] demonstrate that real Optane DCPMMs [5] have more complex performance characteristics than DRAM. Optane DCPMM utilizes 256-byte internal blocks to access data instead of completely byte-addressable access. It has up to about 2.1× higher latency for sequential reads and 3.8× higher latency for random reads compared to DRAM. Moreover, Optane DCPMM has limited scalability compared with DRAM.

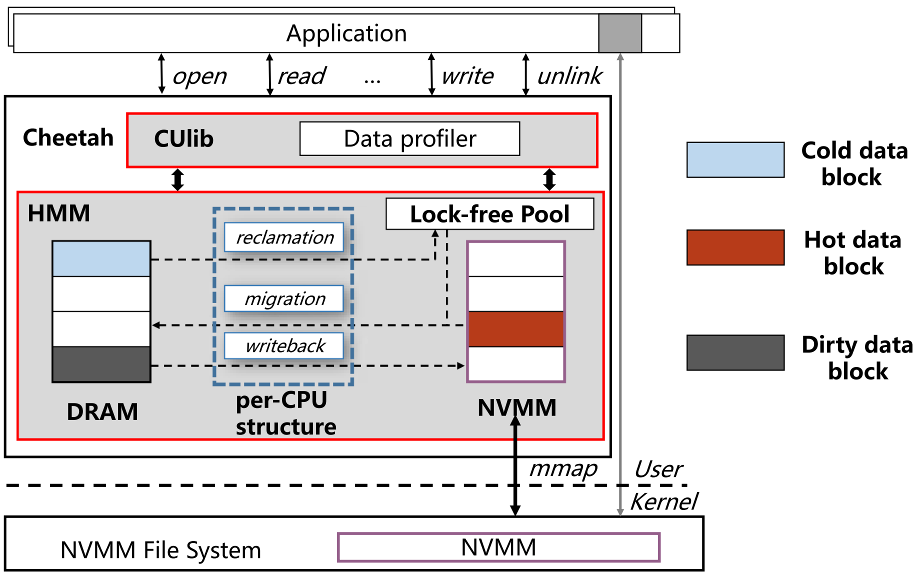

Fig. 1. Cheetah overview.

2.2 Direct Access and Memory Mapping

The Direct Access (DAX) feature of NVMM allows applications to use the CPU `Load/Store` instructions to directly access data in NVMM, bypassing the DRAM page cache. DAX eliminates all operating and file system code from the access path, providing the fastest way to access NVMM. Most NVMM-aware file systems [2,4,7,8,12,16] leverages DAX to shorten the critical path of file I/O. They also support `mmap()` mechanism that directly maps file data in NVMM into user address space. Upon a `mmap()` call, these file systems set up a mapping between file data pages and a range of addresses within an application's address space, reducing context switching and internal addressing overhead.

3 Design

Cheetah is a user-level cache that takes advantage of the benefits of DRAM and NVMM to accelerate file I/O for NVMM file systems. Figure 1 shows the architecture of Cheetah. Cheetah consists of a user-space library and a hybrid memory management module, namely *CUlib* and *HMM*.

CUlib is a user-space library linked to applications. *CUlib* provides common POSIX APIs and contains a *data profiler*. *Data profiler* is a process that profiles the access frequency of file data in order to cache data efficiently. *HMM* is a hybrid memory management module for managing metadata and data both in DRAM and NVMM. To achieve high scalability, HMM manages the hybrid memory space with three groups of threads: *writeback threads*, *reclamation threads*, and *migration threads*. The function of *writeback threads* is to periodically flush dirty data to NVMM for persistence. When the remaining space of the cache is almost used up, the *reclamation* threads evict blocks from cold data. By contrast, the *migration* threads migrate hot file data to DRAM when its temperature reaches a threshold.

We make the following design decisions in Cheetah:

Combined the Benefits of DRAM and NVMM. To hide the long write latency of NVMM, we leverage DRAM to handle asynchronously-updated write requests. Based on the unique features of Optane DCPMMs, we design an adaptive data block allocation policy to avoid write amplification.

Assign Write Destination Adaptively. Different applications have different synchronicity requirements. Cheetah handles asynchronous writes in DRAM and strives other writes to NVMM.

DRAM-Aware Cache Replacement. To obtain high DRAM utilization, Cheetah combines the access frequency of file data and DRAM usage to dynamically adjust the threshold for reclaiming cold data and migrating hot data.

3.1 CUlib

Cheetah provides a user-level library linked to applications called *CUlib* that is fully compatible with existing applications. When the file is opened, Cheetah calls the `mmap()` to map the file data. All interactions between Cheetah and the underlying file systems are based on the mapping regions. We design a module called *data profiler* that identifies what kind of data should be cached in DRAM.

Profiler. Cheetah not only caches hot data in DRAM but also migrates hot data from NVMM to DRAM. *Data profiler* considers both the current free space of DRAM and the access frequency of file data to set the dynamic threshold for identifying hot data. We denote the migration temperature as $T_{migration}$. Once a file's temperature reaches the $T_{migration}$, the migration threads begin to migrate file data to DRAM. T_{total} indicates the total temperature of all data in Cheetah and P_{used} shows the percentage of DRAM space used. The threshold is shown in the following equation:

$$T_{migration} = \begin{cases} T_{total} * 50\% & P_{used} < 50\% \\ T_{total} * P_{used} & 50\% \leq P_{used} < 80\% \\ \infty & 80\% \leq P_{used} < 100\% \end{cases}$$

Writes. *Data profiler* splits file writes into two types: asynchronously-updated writes and synchronously-updated writes. The reason is two-fold. First, persisting all data in NVMM immediately for applications is unnecessary and inefficient. Second, the long write latency of NVMM is always exposed to the critical path, leading to severe system performance degradation. *CUlib* utilizes *DRAM* to absorb asynchronous writes and reduce the high write latency on the critical path, increasing the lifetime of NVMM and improving scalability. For synchronously-updated writes, Cheetah strives it to NVMM directly.

Reads. For read operations with cache misses, the traditional approach has to go through the DRAM cache to fetch file data from disks. However, this approach is not suitable for hybrid DRAM/NVMM architecture. As NVMM and DRAM have similar read performance, Cheetah reads data directly from both DRAM and NVMM to avoid unnecessary data copies in the read path.

3.2 Hybrid Memory Management

HMM is an efficient management module in Cheetah for managing data and metadata on NVMM and DRAM.

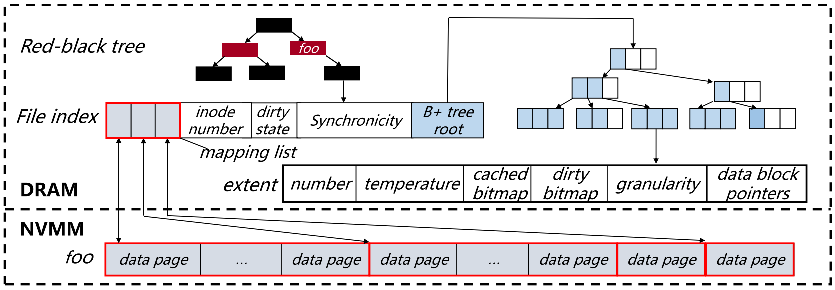

Fig. 2. Data management.

Metadata Management. *HMM* splits the data structures used for managing metadata into per-CPU and lock-free structures to avoid global locking and provide high scalability. Cheetah maintains LRU lists at each CPU to reclaim cold data blocks parallel. Cheetah uses lock-free queues [9] to manage the allocation and deallocation of metadata since it improves the overall throughput by preventing resource contention and context switching.

Fine-Grained Data Block Allocation Policy. Fixed-size data blocks in most file systems are allocated for all writes regardless of its size, which may cause high write amplification. To address the issue, we propose an adaptive data block allocation policy. Cheetah allocates data blocks by the write size. The data block sizes can be set to 256 B, 1 KB, 4 KB, all the way to 1 MB. The minimum block size is 256 B, which is equal to the internal buffer size of Intel Optane DCPMM [17]. Our policy dynamically allocates the most suitable block size according to the I/O size, minimizing write amplification.

Data Management. We utilize a structure called *extent* to manage data blocks. As shown in Fig. 2, each extent contains six fields and manages up to 1 MB data. When the application posts a write request, Cheetah searches the corresponding extents in the B+ tree. If the extents do not exist, new extents are allocated to the file. Reclamation and migration are performed at the granularity of the extent. The advantage of our approach is to make the best of data locality.

File Index. When a file is opened in Cheetah for the first time, Cheetah allocates a unique *index* for the file. Each index records the file metadata (Fig. 2). We build a set of red-black trees to index the files opened in Cheetah, which use the file inode numbers as the key to store the corresponding indexes.

4 Cache Replacement

Cheetah provides an efficient replacement policy to improve space utilization. We next describe the cache replacement policy in detail.

Temperature Profiling. When a data block is accessed, Cheetah increases the temperature of its corresponding extent according to the weight of the access type. The read and write weights are dynamically adjusted when dealing with different types of applications. Cheetah calculates the weight of reads and writes every two seconds, according to the real-time read-write ratio.

Cache Reclamation. Cheetah utilizes background threads to reclaim cold data blocks from the infrequently accessed files. The background threads reclaim the data blocks of different sizes. Once the number of allocated data blocks of any size reaches a threshold (e.g., 90% in our experiments), the reclamation threads wake up and begin to reclaim data blocks from cold *extents*, starting from the tail of LRU lists. We use $T_{threshold}$ (described in Sect. 3.1) as the threshold temperature for cache reclamation. Background threads reclaim all cached data blocks in the extent with temperature lower than $T_{threshold}$ until 20% of the data blocks in the cache are reclaimed.

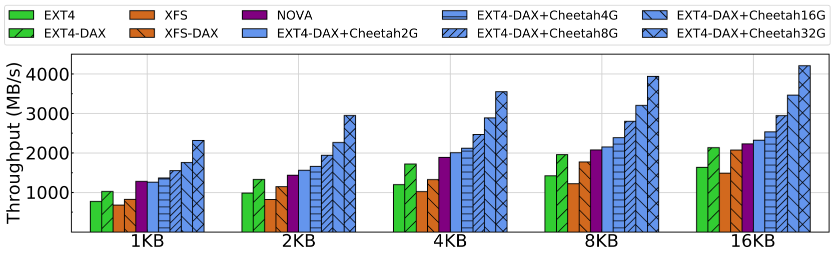

Fig. 3. Fio performance.

Migration. The purpose of migration is to cache more hot data for accelerating the future file I/O. Cheetah sets migration threads at per-CPU and utilizes a lock-free queue as a message buffer to communicate with migration threads. When the *data profiler* identifies an extent with hot temperature, Cheetah pops its index into a lock-free queue, and the migration thread wakes up and migrates its uncached data contained to DRAM.

5 Evaluation

In this section, we evaluate the performance of Cheetah by using a collection of benchmarks and real-world application workload.

5.1 Experimental Setup

Our experimental testbed machine consists of 2× Intel Xeon Gold 6240M CPUs, six 32 GB DDR4 DRAM, and six 256 GB Intel Optane DCPMMs. All the experiments are performed on Ubuntu 18.04 LTS with Linux kernel 4.13.

We use Cheetah-equipped EXT4-DAX (abbreviated as EXT4-DC) as the only Cheetah-equipped file system to compare with other file systems, such as XFS-DAX [4] and NOVA [16], in the FIO (Sect. 5.2) and Redis (Sect. 5.3) workloads. We vary the DRAM cache size available to Cheetah to show how performance changes with different Cheetah configurations. E.g. EXT4-DC (2 GB) means that the DRAM cache size of Cheetah is only 2 GB.

5.2 Microbenchmarks

We use Fio [3] to evaluate the performance of file operations. Fio is a flexible I/O test tool that provides different types of microbenchmarks. We vary the I/O size from 1 KB to 16 KB, and run the workload in single-thread. All I/O operations in this experiment are synchronous.

Figure 3 shows the performance improvement of Cheetah for write operations. EXT4-DC achieves nearly 2.1× higher performance than EXT4 and XFS and outperforms NOVA by 91% and XFS-DAX by 1.5× when the cache capacity is 32 GB. Non-DAX file systems perform poorly on synchronous write operations. Cheetah only absorbs asynchronous writes in DRAM, thus reducing the number of unnecessary double-copies. NOVA utilizes 4 KB pages to manage data. For small writes, this policy causes significant write amplification. Cheetah adopts a fine-grained data allocation policy to minimize write amplification, which allocates the most suitable data blocks instead of fix-sized data blocks.

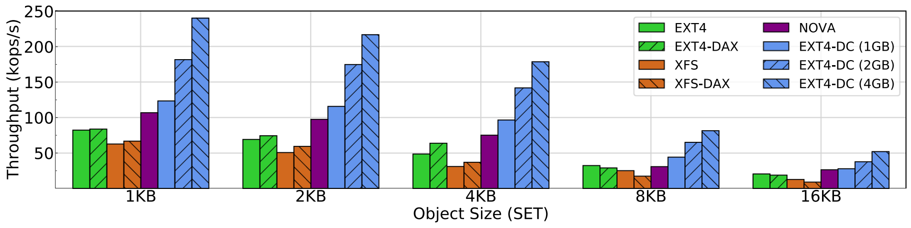

Fig. 4. Redis performance.

5.3 Redis

We use Redis [1] to evaluate the performance of the file systems with Cheetah. We run Redis in the AOF mode, which logs every write operation and persists data synchronously. We vary the object size from 1 KB to 16 KB.

Figure 4 shows the throughput of SET operations of different file systems. EXT4-DC outperforms EXT4-DAX by 51%, 1.2×, and 1.8× on average for the 1 GB, 2 GB, and 4 GB cache capacity, respectively. When the cache capacity is 4 GB, EXT4-DC improves the performance over NOVA by 1.3×, outperforms XFS-DAX, XFS and EXT4 by 3.5×, 3.2×, and 1.9×, respectively.

Compared with EXT4 and XFS, the performance advantage stems primarily from the design of adaptive write services. Compared with the NVMM-aware file systems, the per-CPU and lock-free structures adopted by Cheetah ensure that writing data into DRAM and migrating cold data to NVMM can be performed concurrently to achieve high bandwidth. Also, Cheetah provides direct access without the overhead of context switching and software stack.

6 Related Work

Existing NVMM-aware file systems can be classified into two categories: kernel-space and user-space.

1) *Kernel-space.* EXT4-DAX [2] and XFS-DAX [4] provide direct access capabilities to the native Linux file system. NOVA [16] is designed for hybrid DRAM/NVMM architecture. NOVA guarantees consistency and supports atomic operations through its log-structured design. It stores indexes in DRAM to accelerate lookup and utilizes the logs of each file to achieve high concurrency.

2) *User-Space.* Strata [11] and SplitFS [10] both utilize a split architecture to improve performance. Strata adopts the log digestion scheme to improve file access performance. SplitFS provides flexible guarantees for different applications.

7 Conclusion

We present Cheetah, a user-space cache for NVMM file systems to accelerate file I/O. Cheetah leverages the unique characteristics of Intel Optane DC persistent memory to reduce write amplification. Cheetah absorbs asynchronously-updated writes in DRAM to minimize the impact of long write latency of NVMM. Experimental results show that Cheetah enables applications to exploit the benefits from native file systems while providing significant performance improvements.

Acknowledgments. This work is supported by National Key Research & Development Program of China (Grant No. 2018YFB1003302), and SJTU-Huawei Innovation Research Lab Funding (Grant No. FA2018091021-202004). Shengan Zheng is supported by China Postdoctoral Science Foundation (Grant No. 2020M680570).

References

1. Redis (2010). https://github.com/redis/redis
2. Add support for NV-DIMMs to ext4 (2011). https://lwn.net/Articles/613384/
3. Flexible I/O tester (2011). https://github.com/axboe/fio
4. xfs:DAX support (2015). https://lwn.net/Articles/635514/
5. Intel Optane DC pesistent memory (2019). https://www.intel.com/content/www/us/en/architecture-and-technology/optane-dc-persistent-memory.html
6. Akel, A., Caulfield, A.M., Mollov, T.I., Gupta, R.K., Swanson, S.: Onyx: a prototype phase change memory storage array. In: HotStorage (2011)
7. Condit, J., et al.: Better I/O through byte-addressable, persistent memory. In: SOSP (2009)
8. Dulloor, S.R., et al.: System software for persistent memory (2014)
9. Frechilla, F.: Yet another implementation of a lock-free circular array queue (2011). https://www.codeproject.com/Articles/153898
10. Kadekodi, R., Lee, S.K., Kashyap, S., Kim, T., Kolli, A., Chidambaram, V.: SplitFS: reducing software overhead in file systems for persistent memory. In: SOSP (2019)
11. Kwon, Y., et al.: Strata: a cross media file system. In: SOSP (2017)
12. Ou, J., Shu, J., Lu, Y.: A high performance file system for non-volatile main memory. In: EuroSys (2016)
13. Patil, O., Ionkov, L., Lee, J., Mueller, F., Lang, M.: Performance characterization of a DRAM-NVM hybrid memory architecture for HPC applications using intel Optane DC persistent memory modules. In: Proceedings of the International Symposium on Memory Systems (2019)
14. Waddington, D., Kunitomi, M., Dickey, C., Rao, S., Abboud, A., Tran, J.: Evaluation of intel 3D-xpoint NVDIMM technology for memory-intensive genomic workloads. In: Proceedings of the International Symposium on Memory Systems (2019)
15. Wang, Z., Liu, X., Yang, J., Michailidis, T., Swanson, S., Zhao, J.: Characterizing and modeling non-volatile memory systems. In: MICRO (2020)
16. Xu, J., Swanson, S.: NOVA: a log-structured file system for hybrid volatile/non-volatile main memories. In: FAST (2016)
17. Yang, J., Kim, J., Hoseinzadeh, M., Izraelevitz, J., Swanson, S.: An empirical guide to the behavior and use of scalable persistent memory. In: FAST (2020)