RCache: A Read-Intensive Workload-Aware Page Cache for NVM Filesystem

TU Yaofeng[1,2], ZHU Bohong[3], YANG Hongzhang[1,2], HAN Yinjun[2], SHU Jiwu[3]

(1. State Key Laboratory of Mobile Network and Mobile Multimedia Technology, Shenzhen 518055, China;
2. ZTE Corporation, Shenzhen 518057, China;
3. Tsinghua University, Beijing 100084, China)

DOI: 10.12142/ZTECOM.202301011

https://kns.cnki.net/kcms/detail/34.1294.TN.20230302.1104.002.html, published online March 2, 2023

Manuscript received: 2022-11-01

Abstract: Byte-addressable non-volatile memory (NVM), as a new participant in the storage hierarchy, gives extremely high performance in storage, which forces changes to be made on current filesystem designs. Page cache, once a significant mechanism filling the performance gap between Dynamic Random Access Memory (DRAM) and block devices, is now a liability that heavily hinders the writing performance of NVM filesystems. Therefore state-of-the-art NVM filesystems leverage the direct access (DAX) technology to bypass the page cache entirely. However, the DRAM still provides higher bandwidth than NVM, which prevents skewed read workloads from benefiting from a higher bandwidth of the DRAM and leads to sub-optimal performance for the system. In this paper, we propose RCache, a read-intensive workload-aware page cache for NVM filesystems. Different from traditional caching mechanisms where all reads go through DRAM, RCache uses a tiered page cache design, including assigning DRAM and NVM to hot and cold data separately, and reading data from both sides. To avoid copying data to DRAM in a critical path, RCache migrates data from NVM to DRAM in a background thread. Additionally, RCache manages data in DRAM in a lock-free manner for better latency and scalability. Evaluations on Intel Optane Data Center (DC) Persistent Memory Modules show that, compared with NOVA, RCache achieves 3 times higher bandwidth for read-intensive workloads and introduces little performance loss for write operations.

Keywords: storage system; file system; persistent memory

Citation (IEEE Format): Y. F. Tu, B. H. Zhu, H. Z. Yang, et al., "RCache: a read-intensive workload-aware page cache for NVM filesystem," *ZTE Communications*, vol. 21, no. 1, pp. 89–94, Mar. 2023. doi: 10.12142/ZTECOM.202301011.

1 Introduction

In 2019, Intel released the first commercially available non-volatile memory (NVM) device called Intel DC Optane Persistent Memory[1]. Compared with Dynamic Random Access Memory (DRAM), byte-addressable non-volatile memory provides comparable performance and similar interfaces (e.g., Load/Store) along with data persistence at the same time. Because of a unique combination of features, NVM has a great advantage of performance on storage systems and posts the urgent necessity of reforming the old architecture of storage systems. Refs. [2–11] re-architected the old storage systems to better accommodate NVM and significant performance boost that endorsed these design choices.

Among these novel designs, bypassing the page cache in kernel space is a popular choice. The page cache in Linux is used to be an effective mechanism to shorten the performance gap between DRAM and block devices. Since NVM has a close performance to the DRAM, the page cache itself posts severe performance loss to the NVM filesystem, because the page cache introduces extra data copy at every file operation and leads to write amplification on NVM. Therefore, the legacy page cache in the Linux kernel has become a liability for the NVM system. For the above reasons, recent work simply deployed the DAX[12] technology to bypass the page cache entirely[12–17]. With the DAX technology, NVM filesystems access the address space of NVM directly, without the necessity of filling the page cache first, which reduces the latency of filesystem operations significantly.

However, although NVM achieves bandwidth and latency at the same order of magnitude as DRAM, DRAM still provides bandwidth several times higher than NVM and fairly lower latency than NVM. Therefore, the DAX approach reduces extra data copy and achieves fast write performance at the cost of cached read, especially for read-intensive workloads[18–20]. The page cache provides benefits for reading but has severe performance impacts on writing because of the extra data copy and write amplification. And the DAX approach is efficient for writing due to direct access to NVM but fails to utilize DRAM bandwidth for reading. Therefore, in order to utilize DRAM

This paper was supported by ZTE Industry-University-Institute Cooperation Funds under Grant No. HC-CN-20181128026.

bandwidth and avoid extra data copy and write amplifications, the page cache should be redesigned to allow both direct access and cached read.

In this paper, we propose RCache, a read-intensive workload-aware page cache for the NVM filesystem. RCache aims to provide fast read performance for read-intensive workloads and avoid introducing significant performance loss for write operations at the same time. To achieve this, RCache assigns DRAM and NVM to hot and cold data separately, and reads data from both sides. Our major contributions are summarized as follows.

• We propose a read-intensive workload-aware page cache design for the NVM filesystem. RCache uses a tired page cache design, including reading hot data from DRAM and accessing cold data directly from NVM to utilize DRAM bandwidth for reading and preserving fast write performance. In addition, RCache offloads data copy from NVM to DRAM and to a background thread, in order to remove a major setback of caching mechanism from the critical path.

• RCache introduces a hash-based page cache design to manage the page cache in a lock-free manner using atomic instructions for better scalability.

• We implement RCache and evaluate it on servers with Intel DC Persistent Memory Modules. Experimental results show that RCache effectively utilizes the bandwidth of DRAM with few performance cost to manage the page cache and outperforms the state-of-the-art DAX filesystem under read-intensive workloads.

2 Background and Motivation

2.1 Non-Volatile Memory

Byte-addressable NVM technologies, including Phase-change Memory (PCM) [22-24], ReRAM, and Memristor[21], have been intensively studied in recent years. These NVMs provide comparable performance and a similar interface as the DRAM, while persisting data after power is off like block devices. Therefore, NVMs are promising candidates for providing persistent storage ability at the main memory level. Recently, Intel has released Optane DC Persistent Memory Modules (DCPMM)[1], which is the first commercially available persistent memory product. Currently, new products come in three capacities: 128 GB, 256 GB, and 512 GB. Previous studies show that a single DCPMM provides bandwidths at 6.6 GB/s and 2.3 GB/s at most for read/write. Note that these bandwidth have the same order of magnitudes comparable to the DRAM but is a lot lower than the DRAM[25].

2.2 Page Cache and DAX Filesystem

Page cache is an important component in a Linux kernel filesystem. In brief, the page cache consists of a bunch of pages in DRAM and the corresponding metadata structures. The page cache is only accessed by the operating system in the context of a filesystem call and acts as a transparent layer to user applications. For a write system call, the operating system writes data on pages in the page cache, which cannot guarantee the persistence of the data. To guarantee the persistence of the data, the operating system needs to flush all data pages in the page cache to the storage devices, probably within an fsync system call. For a read system call, the operating system first reads data from the page cache; if not present, the operating system further reads data from the storage devices. Note that this may involve loading data into the page cache depending on the implementation. In the current implementation, the operating system maintains an individual radix tree for each opened file.

As for the DAX filesystem, note that the page cache is extremely useful for block devices with much higher access latency than DRAM, but not suitable for the NVM devices with comparable access latency to DRAM. As mentioned before, to ensure data persistence, the user must issue an fsync system call after a write system call. This brings substantial access latency to persisting data in an NVM filesystem. Therefore, the state-of-the-art NVM filesystems leverage the DAX technology to bypass the page cache entirely and achieve instant persistence immediately when the write system call returns. In a DAX filesystem, read/write system call does not access the page cache at all, instead, data are loaded/stored from/to the NVM respectively using a memory interface. The DAX technology reduces extra data copy and accomplishes lower-cost data persistence.

2.3 Issue of DAX and Page Cache

The performance of NVM is close to that of DRAM but not equal to it. We measure the read and write latency of two different filesystems (NOVA[17] and EXT4[26]) representing two different mechanisms (DAX and Page Cache). Fig. 1(a) shows that the read latency of the DAX is much higher than the page cache (4 kB sequential read). Fig. 1(b) shows that the write latency of the DAX is much lower than the page cache (4 kB sequential write).

To sum up, the DAX technology prevents the read operations from benefiting a much higher bandwidth of DRAM in the NVM filesystem, and the presence of the page cache significantly increases the latency of write operations with immediate data persistence. To overcome this, the page cache mechanism needs to be redesigned.

3 Rcache Design

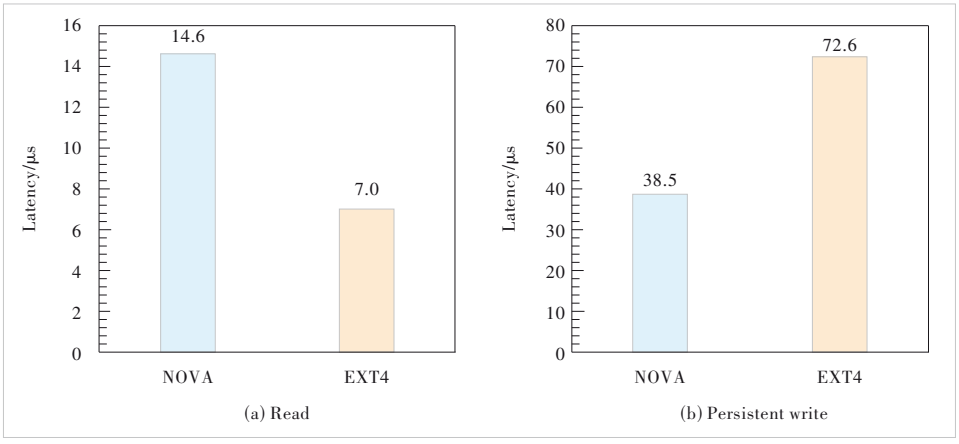
3.1 Overview

We build RCache for servers with non-volatile memory to accelerate read-intensive workloads. In order to benefit from the DRAM bandwidth for read operations but not to induce notable latency for data persistence, we build RCache, a read-intensive workload-aware page cache for the NVM filesystem.

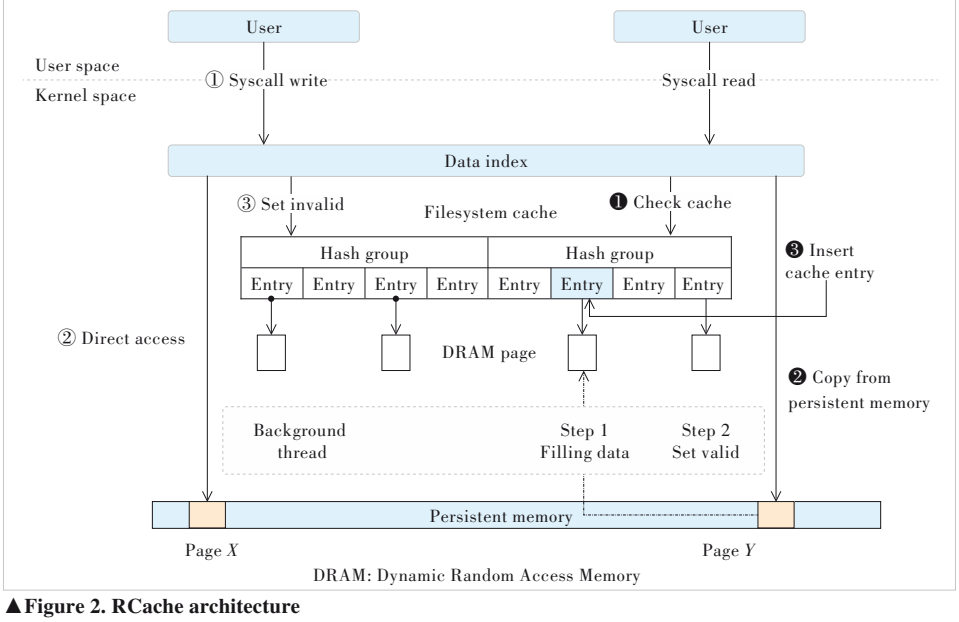
1) RCache assigns DRAM and NVM to hot and cold data separately, and allows cached read and direct read from NVM to coexist. Furthermore, RCache offloads data copy to a background thread to alleviate the pressure of the critical path.

2) In addition, RCache deploys a lock-free page cache using hash-table to further reduce the performance cost of cache coherence management.

The architecture of RCache is described in Fig. 2. RCache keeps an individual cache structure for each opened file. The page cache consists of a bunch of DRAM pages and a cache entry table containing a certain number of cache entries in the DRAM. A cache entry represents a DRAM page. It carries necessary information for RCache to manage the cache and navigate data given a logical block number. As shown in Fig. 3, a cache entry carries a validation flag to indicate the status of this cache entry, a timestamp for the least recently used (LRU) algorithm, a Blocknr to indicate the logical block number that the entry represents, a DRAM page that is a pointer points to the actual cache page in DRAM, and an NVM page that is a pointer points to the actual data page in NVM.



▲Figure 1. Performance comparison between different hardware and different filesystem settings



▲Figure 2. RCache architecture

3.2 Tiered Page Cache Design

As shown in Fig. 2, the page cache is accessed in two contexts: a read/write system call and a background thread.

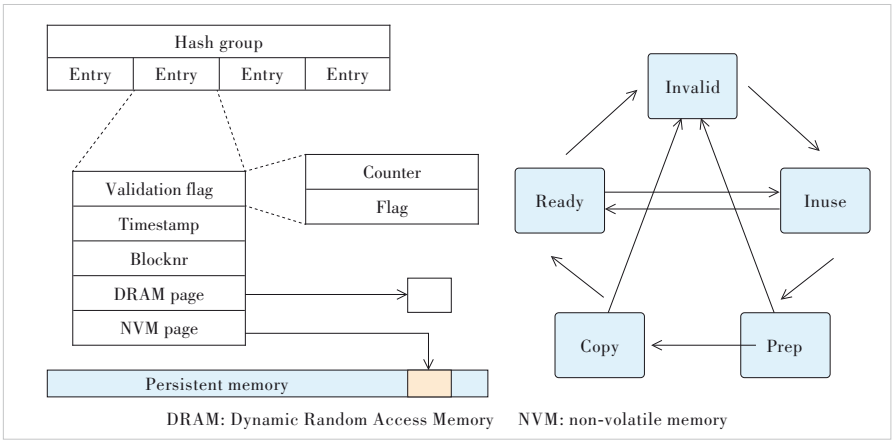
For a read operation, the operating system accesses the page cache first. If the data required by the user are present and valid in the page cache, the operating system copies data directly from the cached page in the DRAM to the user's buffer; if a cache miss happens, the operating system falls back to the legacy procedure where the operating system reads data directly from the NVM and inserts the newly read data to the page cache. For cache insertion, since reading all the data blocks into the page cache introduces extra data copy and then leads to higher latency, RCache only inserts a small cache entry carrying a pointer to the physical block to the page cache instead of the actual data blocks.

For a write operation, the operating system needs to invalidate all cached pages affected by this write operation before returned to users. We further explain why the invalidation procedure is light weight in Section 3.3.

RCache depends on a background kernel thread to finish the management of the cache. As described above, in the read operation, RCache only inserts cache entries to the page cache. In the context background thread, once a pending cache entry is discovered, RCache first allocates a DRAM page to cache data, and then copies data from the NVM block to the DRAM page according to the cache entry. At last, RCache declares the validity of the cache entry by switching the validation flag atomically. Note that only when RCache updates the validation flag in the cache entry to validation, the cache entry is available for read/write context.

3.3 Lock-Free Cache Management

The decoupled cache mechanism splits the cache management into two separating and concurrent contexts, which makes coordinating across all units more expensive since it leads to more cross-core communications. Therefore, RCache



▲Figure 3. Cache structure and status shifting paradigm

deploys a lock-free cache management procedure to minimize the impact. First, RCache operates cache entries by manipulating the validation flag atomicity using Compare-and-Swap (CAS) instructions. In the current implementation, a cache entry switches among five states using the Compare-and-Swap instruction. Fig. 3 depicts the transition diagram among these five states. At the initial point, all cache entries are invalid. To insert a cache entry, RCache first acquires control of a candidate entry by setting the validation flag of this entry to "In use" atomically using CAS, which prevents other threads from operating on this entry. Then, RCache fills necessary information (e.g. the block number and the NVM page pointer) and changes the status to "Prep", which tells the background thread that this entry has all information needed and is ready for data copy. From the background thread view, before copying data from persistent memory to DRAM, the background thread first sets the status of a cache entry to "Copy", then the background thread initiates a data copy procedure. When the data copy completes, the background thread sets the status of a cache entry to "Ready" by using CAS instruction operating on the validation flag, and, only at this point, the cache is available for read operations. To write data into a certain page, if cache hits, RCache needs to invalidate the cache entry representing this page by switching the status to "Invalid" by CAS, and the validation flag of the entry to "Invalid". Note that RCache never invalidates an "In use" cache entry, because the "In use" status only exists in the context of a read syscall. Since the file is locked up in write operations, this situation never happens. To read data from a cache entry, RCache first switches the status from "Ready" to "In use" using CAS, then copies data from the DRAM page to user buffer, and at last, changes the status back to "Ready". However, this leads to an inconsistent status where users might be given wrong data, since there might be several threads reading data from the cache entry concurrently. Therefore, RCache incarnates an additional counter in the validation flag, when a reader wants to read this cache, it must increase this counter; and when a reader finishes reading, it must decrease the counter. Therefore, only the last reader can switch the status back to "Ready".

3.4 Implementation

We implement RCache on NOVA, a state-of-the-art NVM filesystem developed with the DAX technology. We keep the metadata and data layout in NOVA intact, and add extra logic for managing the cache in the context of read/write procedure. We launch the background thread in kernel at the mount phase, and reclaim this thread during the unmount phase. To tackle the hotness of a block, we extend the block index in NOVA, and add an extra counter to each leaf node of the radix tree. We insert a block into the cache only when it is accessed more times than a threshold in a time window. The threshold and the time window are predefined.

4 Evaluation

In this section, we first evaluate RCache's read/write latency, then we evaluate the read performance under read-intensive workload, and at last, we evaluate the read performance under a skewed read-intensive workload.

4.1 Experimental Setup

We implement RCache and evaluate the performance of RCache on the server with Intel Optane DCPMM. The server has 192 GB DRAM and two Intel Xeon Gold 6 240 M processors (2.6 GHz, 36 cores per processor) and 1 536 GB Intel Optane DC Persistent Memory Modules (6×256 GB). Because cross-non-uniform memory access (NUMA) traffic has a huge impact on performance[27], throughout the entire evaluation, we only utilize NVMs on one NUMA node to deploy RCache and other file systems (e.g., only 768 GB NVMs on this server). The server is running Ubuntu18.04 with Linux Kernel 4.15.

Table 1 lists file systems for comparison. We build all filesystems on the same NVM device with a PMem driver. For EXT4, we build it following the traditional procedure with a page cache involved. For both NOVA and RCache, since RCache shares most of the filesystem routines with NOVA, we deploy both of them on an NVM device with a PMem driver and DAX enabled.

For a latency test, we use custom micro benchmarks and Fxmark[28] for bandwidth evaluation. Fxmark is a benchmark de-

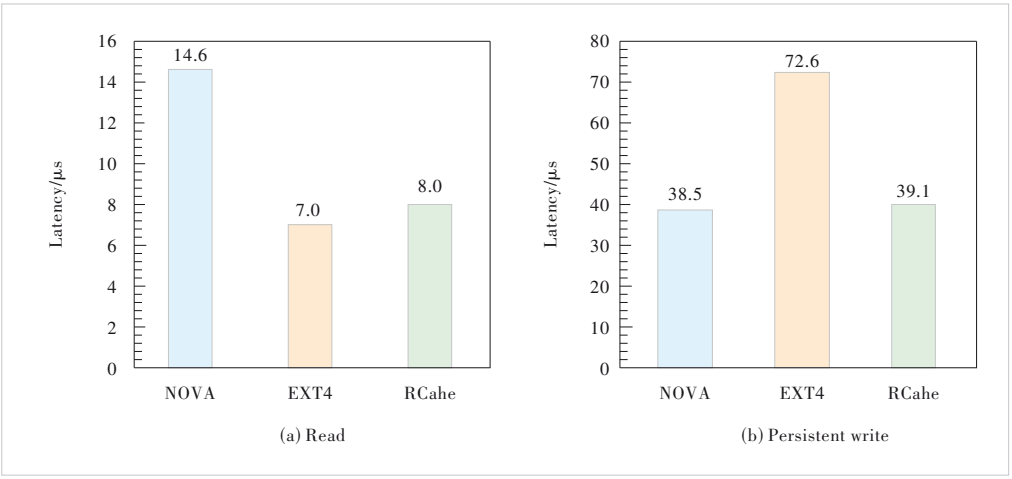
▼Table 1. Evaluated file systems

File System	Description
NOVA[17]	A state-of-the-art NVM filesystem in the kernel. NOVA adopts conventional log-structured file system techniques and optimizes file systems for hybrid memory systems to maximize performance
EXT-4[26]	A well-known kernel file system in Linux

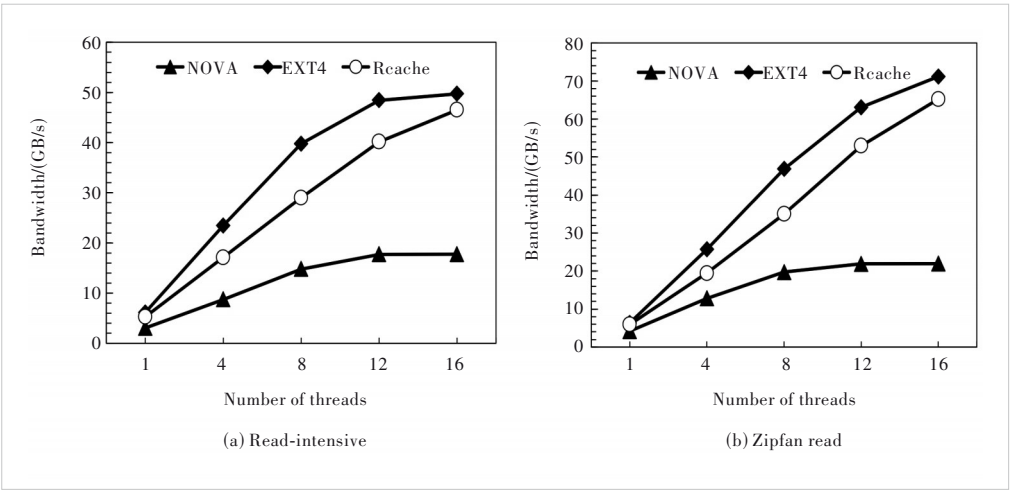
signed to evaluate the scalability of file systems. In this evaluation, we use three sub-benchmarks, namely DRBL, DRBM and DWAL, in Fxmark.

4.2 Overall Performance

To evaluate the read/write performance, we use a custom micro-benchmark. All evaluation on each filesystem spawns only one thread. We first create a file with 64 MB, then issue 4 kB read/write data with 100 000 requests, and finally calculate the average latency. Since EXT4 does not ensure data persistency in the write system call, we issue another fsync after each write system call to preserve data persistency. Fig. 4 shows the read/write latency for three evaluated filesystems.



▲Figure 4. Read and write latency of different filesystems



▲Figure 5. Read bandwidth under the read-intensive workload of different filesystems

For read operations, EXT4 shows the lowest latency, and the latency of RCache is close to that of EXT4 and much lower than that of NOVA. This is because RCache utilizes the DRAM bandwidth to accelerate read.

To evaluate the read bandwidth under a read-intensive workload, we use sub-benchmark DRBL from Fxmark. DRBL first creates a 64 MB file for each thread and then issues sequence read operation to the filesystem. We conduct the evaluation for 20 s. If a read operation reaches the tail of the file, the next read operation is set at the beginning of the file. From Fig. 5(a) we can see that the RCache shows much better read performance than NOVA and close to that of EXT4.

4.3 Read Performance Under Skewness

We evaluate the read performance under the skewed workload. We modify the DRBL benchmark instead of reading files sequentially, where each thread post-read request at an offset is controlled by a random variable that follows the normal distribution. Fig. 5(b) shows that, both EXT4 and RCache achieve even better performance than that in Fig. 5(a). This is because under the skewed workload, the hot pages are more likely to be stored in the L3 cache and therefore end up with better performance. On the other hand, since NOVA does not utilize DRAM for better read performance, the read bandwidth achieved is much lower than that of EXT4 or RCache.

5 Conclusions

Traditional page cache in the Linux kernel can benefit read workload but cannot fit into an NVM filesystem because it causes extra data copy and write amplification. By bypassing the page cache, the DAX filesystem achieves better write performance but gives up the opportunity of cached read. Therefore, in this paper, we propose a read-intensive workload-aware page cache for NVM filesystems. RCache uses a tiered page cache design, including assigning DRAM and NVM to hot and cold data separately, and reading data from both sides. Therefore, cached read and direct access can coexist. In addition, to avoid copying data to DRAM in a critical path, RCache migrates data from NVM to DRAM in a background thread. Furthermore, RCache manages data in DRAM in a lock-free manner for better latency and scalability. Evaluations on Intel Optane DC Persistent Memory Modules show that compared with NOVA, RCache has 3 times higher bandwidth for read-intensive workloads and introduces little performance loss to write operations.

References

[1] Intel. Intel Optane DC Persistent Memory [EB/OL]. [2022-11-01]. https://www.intel. com/content/www/us/en/products/memory-storage/optane-dcpersistent-memory.html
[2] SHU J W, CHEN Y M, WANG Q, et al. TH-DPMS: design and implementation of an RDMA-enabled distributed persistent memory storage system [J]. ACM transactions on storage, 2020, 16(4): 1 - 31. DOI: 10.1145/3412852
[3] CHEN Y M, LU Y Y, SHU J W. Scalable RDMA RPC on reliable connection with efficient resource sharing [C]//Proceedings of the Fourteenth EuroSys Conference. ACM, 2019. DOI: 10.1145/3302424.3303968
[4] CHEN Y M, LU Y Y, YANG F, et al. FlatStore: an efficient log-structured key-value storage engine for persistent memory [C]//Proceedings of the Twenty-Fifth International Conference on Architectural Support for Programming Languages and Operating Systems. ACM, 2020: 1077 - 1091. DOI: 10.1145/3373376.3378515
[5] LU Y Y, SHU J W, CHEN Y M, et al. Octopus: an RDMA-enabled Distribute Persistent Memory File System [C]//USENIX Annual Technical Conference. IEEE, 2017: 773 - 785
[6] ZHU B H, CHEN Y M, WANG Q, et al. Octopus+: an RDMA-Enabled Distributed Persistent Memory File System [J]. ACM Transactions on Storage, 2021, 17 (3): 1 - 25
[7] COBURN J, CAULFIELD A M, AKEL A, et al. NV-Heaps: making persistent objects fast and safe with next-generation, non-volatile memories [C]//Proceedings of the 16th International Conference on Architectural Support for Programming Languages and Operating Systems. ACM, 2011: 105 - 118. DOI: 10.1145/1950365.1950380
[8] HONDA M, EGGERT L, SANTRY D. PASTE: network stacks must integrate with NVMM abstractions [C]//Proceedings of the 15th ACM Workshop on Hot Topics in Networks. ACM, 2016: 183 - 189. DOI: 10.1145/3005745.3005761
[9] NARAYANAN D, HODSON O. Whole-system persistence [C]//Proceedings of the 17th International Conference on Architectural Support for Programming Languages and Operating Systems. ACM, 2012: 401 - 410. DOI: 10.1145/2150976.2151018
[10] VOLOS H, TACK A J, SWIFT M M. Mnemosyne: lightweight persistent memory [C]//Proceedings of the 16th International Conference on Architectural Support for Programming Languages and Operating Systems. ACM, 2011: 91 - 104. DOI: 10.1145/1950365.1950379
[11] ZHANG Y Y, YANG J, MEMARIPOUR A, et al. Mojim: a reliable and highly-available non-volatile memory system [C]//Proceedings of the 20th International Conference on Architectural Support for Programming Languages and Operating Systems. New York: ACM, 2015: 3 - 18. DOI: 10.1145/2694344.2694370
[12] DULLOOR S R, KUMAR S, KESHAVAMURTHY A, et al. System software for persistent memory [C]//Proceedings of the 9th European Conference on Computer Systems. ACM, 2014: 15 - 30. DOI: 10.1145/2592798.2592814
[13] CHEN Y M, LU Y Y, ZHU B H, et al. 2021. Scalable Persistent Memory File System with Kernel-Userspace Collaboration [C]//USENIX Conference on File and Storage Technologies (FAST 21). FAST, 2021: 81 - 95
[14] DONG M K, BU H, YI J F, et al. Performance and protection in the ZoFS user-space NVM file system [C]//Proceedings of the 27th ACM Symposium on Operating Systems Principles. ACM, 2019: 478 - 493. DOI: 10.1145/3341301.3359637
[15] KADEKODI R, LEE S K, KASHYAP S, et al. SplitFS: reducing software overhead in file systems for persistent memory [C]//Proceedings of the 27th ACM Symposium on Operating Systems Principles. ACM, 2019: 494 - 508. DOI: 10.1145/3341301.3359631
[16] OU J X, SHU J W, LU Y Y. A high performance file system for non-volatile main memory [C]//Proceedings of the Eleventh European Conference on Computer Systems. ACM, 2016: 1 - 16 DOI: 10.1145/2901318.2901324
[17] XU J, SWANSON S. NOVA: a log-structured file system for hybrid volatile/non-volatile main memories [C]//The 14th USENIX Conference on File and Storage Technologies. ACM, 2016: 323 - 338. DOI: 10.5555/2930583.2930608
[18] ATIKOGLU B, XU Y H, FRACHTENBERG E, et al. Workload analysis of a large-scale key-value store [C]//Proceedings of the 12th ACM SIGMETRICS/PERFORMANCE Joint International Conference on Measurement and Modeling of Computer Systems. ACM, 2012: 53 - 64. DOI: 10.1145/2254756.2254766
[19] LI J L, NELSON J, MICHAEL E, et al. Pegasus: tolerating skewed workloads in distributed storage with in-network coherence directories [C]//Proceedings of the 14th USENIX Conference on Operating Systems Design and Implementation. ACM, 2020: 387 - 406. DOI: 10.5555/3488766.3488788
[20] YANG J C, YUE Y, RASHMI K V. A large scale analysis of hundreds of in-memory cache clusters at Twitter [C]//The 14th USENIX Symposium on Operating Systems Design and Implementation (OSDI 20). ACM, 2014: 191 - 208
[21] STRUKOV D B, SNIDER G S, STEWART D R, et al. The missing memristor found [J]. Nature, 2008, 453(7191): 80 - 83. DOI: 10.1038/nature06932
[22] LEE B C, IPEK E, MUTLU O, et al. Architecting phase change memory as a scalable dram alternative [C]//Proceedings of the 36th Annual International Symposium on Computer Architecture. ACM, 2009: 2 - 13. DOI: 10.1145/1555754.1555758
[23] QURESHI M K, SRINIVASAN V, RIVERS J A. Scalable high performance main memory system using phase-change memory technology [C]//Proceedings of the 36th Annual International Symposium on Computer Architecture. ACM, 2009: 24 - 33. DOI: 10.1145/1555754.1555760
[24] ZHOU P, ZHAO B, YANG J, et al. A durable and energy efficient main memory using phase change memory technology [C]//Proceedings of the 36th Annual International Symposium on Computer Architecture. ACM, 2009: 14 - 23. DOI: 10.1145/1555754.1555759
[25] IZRAELEVITZ J, YANG J, ZHANG L, et al. Basic performance measurements of the intel optane DC persistent memory module [EB/OL]. [2022-03-14]. https://arxiv.org/abs/1903.05714
[26] EXT4. EXT4 (and EXT2/EXT3) Wiki [EB/OL]. (2016-09-20) [2022-03-14]. https://ext4.wiki.kernel.org/
[27] YANG J, KIM J, HOSEINZADEH M, et al. An empirical guide to the behavior and use of scalable persistent memory [C]//The 18th Conference on File and Storage Technologies. ACM, 2020: 169 - 182
[28] MIN C W, KASHYAP S, MAASS S, et al. Understanding manycore scalability of file systems [C]//USENIX Annual Technical Conference. ACM, 2016: 71 - 85

Biographies

TU Yaofeng received his PhD degree from Nanjing University of Aeronautics and Astronautics, China. He is a senior expert at ZTE Corporation. His research interests include big data, database and machine learning.

ZHU Bohong (zhubohong18@mails.tsinghua.edu.cn) received his master's degree from Tsinghua University, China in 2018. He is currently studying in the School of Informatics, Xiamen University, China for his PhD degree. His research interests include filesystems, memory storage and distributed systems.

YANG Hongzhang received his PhD degree from Peking University, China. He is an engineer at ZTE Corporation. His research interests include file systems, persistent memory and storage reliability.

HAN Yinjun received his master's degree from Nanjing University of Science and Technology, China. He is a senior engineer at ZTE Corporation. His research interests include distributed file systems, RDMA and persistent memory.

SHU Jiwu received his PhD degree in computer science from Nanjing University, China in 1998. He is currently the dean of the School of Informatics, Xiamen University, China and a professor in the Department of Computer Science and Technology, Tsinghua University, China. His research interests include network storage systems, non-volatile memory systems and technologies, reliability for storage systems, and parallel/distributed processing technologies. He is a Changjiang Professor, CCF Fellow, and IEEE Fellow.