POSTER: AR-MMAP: Write Performance Improvement of Memory-Mapped File

Satoshi Imamura and Eiji Yoshida
ICT Systems Laboratory, Fujitsu Laboratories Ltd., Kasawaki, Japan
{s-imamura, yoshida.eiji-01}@fujitsu.com

I. Introduction

Main memory is one of the most important components on modern computer systems, and various applications require a larger memory capacity for handling the explosively increasing amount of data. [1]–[4]. DRAM has been commonly used as main memory for a few decades, but its density and cost are not expected to scale further due to physical limitations [5]. On the other hand, non-volatile memory used in block storage devices (e.g., NAND flash memory and Intel® 3D XPoint™ memory used in SSDs) still continue to scale because of technology improvements [6]. Therefore, *hybrid memory systems* that use block devices as an extension of DRAM are attracting a lot of attention in both industry and academia, because they can configure high-performance, large-capacity, and low-cost main memory [7]–[13].

II. Motivation

A memory-mapped file is a virtual memory feature supported by various operating systems, which can be used to configure hybrid memory systems. It enables applications to transparently access data stored on block devices by mapping files to their virtual memory address spaces. When an application accesses data not stored in DRAM, a CPU raises a page fault and an operating system synchronously reads a corresponding page from a mapped file on a block device and store it in a DRAM cache called *page cache*. After that, the CPU performs a load/store instruction to access the data.

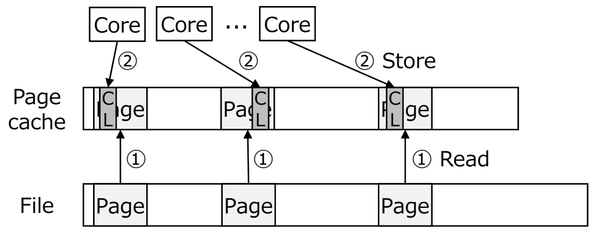
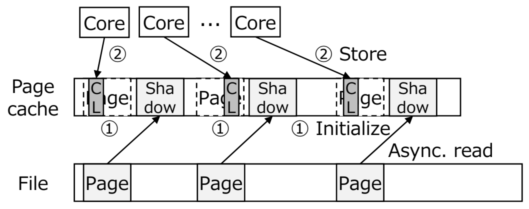
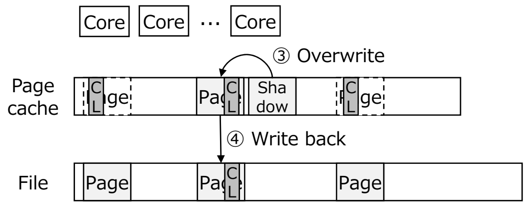


Fig. 1. Store-after-Read (SaR) operations to update data that is not stored in the page cache.

Unfortunately, even when an application updates a portion of a page that is not stored in the page cache, an operating system must synchronously read the entire page from a file due to the data management in a page granularity. In this case, as the application writes new data to the page, it does not require old data stored in the file. In this paper, we call this procedure *Store-after-Read (SaR) operation* and illustrate it in Fig. 1. It puts the read latency of a block device in the critical path of a store instruction.



(a) Write page fault handling



(b) Write back operation in background

Fig. 2. Design of AR-MMAP-PF.

III. AR-MMAP

Therefore, we propose *AR-MMAP* that is an asynchronous read method to reduce the impact of SaR operations on application performance. We illustrate the design of AR-MMAP in Fig. 2. When a write page fault occurs, AR-MMAP initializes every cache line in a new page allocated in the page cache to a specific *initial value* and completes page fault handling without reading a corresponding page from a file. Instead, it reads the page from a file in background. Thus, a CPU can perform a store instruction immediately without waiting the completion of a read I/O operation. When a dirty page in the page cache is written back to a file, AR-MMAP identifies cache lines that are not updated by an application and overwrites them with the correct data obtained from a file.

However, AR-MMAP has three requirements for applications to guarantee data consistency. First, applications must write data to a memory-mapped region in units of cache lines. If a cache line is partially updated, some portions of an initial value written by AR-MMAP are written back to a mapped file. Second, applications must detect reads of an initial value and synchronize corresponding pages in the page cache with correct data stored in mapped files. Otherwise,

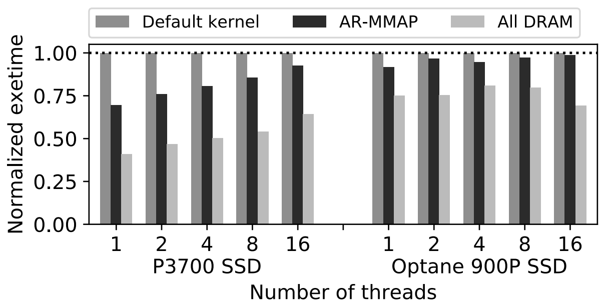


Fig. 3. Execution time evaluation with the YCSB running on memcached.

initial values in the page cache might be read by applications. Third, applications must not write data that is identical to an initial value to a memory-mapped region, because AR-MMAP overwrites cache lines containing initial values with old data. Although it is not trivial to modify applications to satisfy these requirements, AR-MMAP does not require any additional hardware owing to the assistance of applications.

IV. Evaluation

We implement AR-MMAP in a Linux kernel version 4.14.84 and compare its performance with that of a default kernel on a real server that contains an 18-core Xeon® E5-2697 v4 processor, Intel® DC P3700 SSD [14], and Intel® Optane™ 900P SSD [15]. In this paper, we use *memcached* [16], an in-memory key-value cache, as a use case and modify it so that it satisfies all of the three requirements of AR-MMAP. For performance evaluation of AR-MMAP, we run the Yahoo! Could Serving Benchmark (YCSB) [17] on the modified memcached. We configure each thread of this benchmark to process 100,000 1 KB records and vary the number of threads from one to sixteen. Note that in order to emulate a situation where the total data size of the YCSB is larger than the capacity of DRAM, the size of page cache is limited to 25% of the total data size with the Linux *cgroup*.

Fig. 3 plots the execution time of a read-update mixed (50% reads and 50% updates) workload of the YCSB. The execution time is normalized by that measured with the default kernel for each number of threads. In this figure, we can see that AR-MMAP reduces the execution time by 7.3% to 30.5% with the P3700 SSD compared to the default kernel. The benefit of AR-MMAP gets smaller as the number of threads is increased, because the impact of SaR operations can be hidden with the parallelism of more threads. This figure also includes the results of *All DRAM* that stores all records of the YCSB in DRAM. As all read operations in addition to update operations are also performed on DRAM in this case, the execution time reductions are larger than those by AR-MMAP. On the other hand, the benefit of AR-MMAP is very small with the Optane 900P SSD, because its performance is much higher than that of the P3700 SSD. However, as its cost is also much higher than that of the P3700 SSD, AR-MMAP is effective when we configure a low-cost hybrid memory systems with slower but cheaper SSDs like the P3700 SSD.

V. Related Work

Various techniques have ever been proposed to improve the performance and/or usability of memory-mapped files [8]–[13]. However, they cannot alleviate the impact of SaR operations on application performance. Papagiannis et al. [18] implemented a custom memory-mapped I/O mechanism in their original key-value store. It avoids unnecessary SaR operations for pages that are entirely updated. In contrast, our proposed AR-MMAP can alleviate the impact of SaR operations even for pages that are partially updated.

References

[1] N. Mukherjee, S. Chavan, M. Colgan, D. Das, M. Gleeson, S. Hase, A. Holloway, H. Jin, J. Kamp, K. Kulkarni, T. Lahiri, J. Loaiza, N. Macnaughton, V. Marwah, A. Mullick, A. Witkowski, J. Yan, and M. Zait, "Distributed Architecture of Oracle Database In-memory," *Proc. VLDB Endow.*, vol. 8, no. 12, pp. 1630–1641, 2015.

[2] Z. Caklovic and O. Rebholz, "Bringing Persistent Memory Technology to SAP HANA : Opportunities and Challenges," 2017, SNIA Persistent Memory Summit.

[3] J. Ahn, S. Hong, S. Yoo, O. Mutlu, and K. Choi, "A Scalable Processing-in-memory Accelerator for Parallel Graph Processing," ser. ISCA '15, 2015, pp. 105–117.

[4] Y. Kwon and M. Rhu, "Beyond the Memory Wall: A Case for Memory-Centric HPC System for Deep Learning," ser. MICRO 51, 2018, pp. 148–161.

[5] S. R. Dulloor, A. Roy, Z. Zhao, N. Sundaram, N. Satish, R. Sankaran, J. Jackson, and K. Schwan, "Data Tiering in Heterogeneous Memory Systems," ser. EuroSys '16, 2016, pp. 15:1–15:16.

[6] J. Yoon, R. Godse, and A. Walls, "3D NAND Technology Scaling helps accelerate AI growth," 2018, Flash Memory Summit.

[7] F. T. Hady, A. Foong, B. Veal, and D. Williams, "Platform Storage Performance With 3D XPoint Technology," *Proceedings of the IEEE*, vol. 105, no. 9, pp. 1822–1833, 2017.

[8] C. Wang, S. S. Vazhkudai, X. Ma, F. Meng, Y. Kim, and C. Engelmann, "NVMalloc: Exposing an Aggregate SSD Store as a Memory Partition in Extreme-Scale Machines," ser. IPDPS '12, 2012, pp. 957–968.

[9] J. Huang, A. Badam, M. K. Qureshi, and K. Schwan, "Unified Address Translation for Memory-mapped SSDs with FlashMap," ser. ISCA '15, 2015, pp. 580–591.

[10] N. Y. Song, Y. Son, H. Han, and H. Y. Yeom, "Efficient Memory-Mapped I/O on Fast Storage Device," *ACM Trans. Storage*, vol. 12, no. 4, pp. 19:1–19:27, 2016.

[11] B. Essen, H. Hsieh, S. Ames, R. Pearce, and M. Gokhale, "DI-MMAP–a Scalable Memory-map Runtime for Out-of-core Data-intensive Applications," *Cluster Computing*, vol. 18, no. 1, pp. 15–28, 2015.

[12] L. Wang, Q. Wang, L. Chen, and X. Hao, "Fine-Grained Data Management for DRAM/SSD Hybrid Main Memory Architecture," *IEICE Transactions on Information and Systems*, vol. E99D, no. 12, pp. 3172–3176, 2016.

[13] A. Badam and V. S. Pai, "SSDAlloc: Hybrid SSD/RAM Memory Management Made Easy," ser. NSDI'11, 2011, pp. 211–224.

[14] "Intel® SSD DC P3700 Series," https://ark.intel.com/products/79620/Intel-SSD-DC-P3700-Series-2_0TB-12-Height-PCIe-3_0-20nm-MLC, Last accessed: April, 2019.

[15] "Intel® Optane™ SSD 900P Series," https://ark.intel.com/products/123626/Intel-Optane-SSD-900P-Series-480GB-12-Height-PCIe-x4-20nm-3D-XPoint, Last accessed: April, 2019.

[16] B. Fitzpatrick, "Distributed Caching with Memcached," *Linux Journal*, vol. 2004, no. 124, p. 5, 2004.

[17] B. F. Cooper, A. Silberstein, E. Tam, R. Ramakrishnan, and R. Sears, "Benchmarking Cloud Serving Systems with YCSB," ser. SoCC '10, 2010, pp. 143–154.

[18] A. Papagiannis, G. Saloustros, P. González-Férez, and A. Bilas, "An Efficient Memory-Mapped Key-Value Store for Flash Storage," ser. SoCC '18, 2018, pp. 490–502.