POSTER: Understand the Overheads of Storage Data Structures on Persistent Memory

Abdullah Al Raqibul Islam
aislam6@uncc.edu
University of North Carolina at Charlotte
Charlotte, NC

Dong Dai
ddai@uncc.edu
University of North Carolina at Charlotte
Charlotte, NC

Abstract

The byte-addressable persistent memory (PMEM) devices have opened new opportunities for building high-performance storage systems. With both DRAM and PMEM in the system, it is important to choose the correct storage data structures on each of them to achieve the best overall performance and the needed data persistence. However, this is non-trivial. One reason is the limited understanding of the actual performance characteristic of different data structures on PMEM. In this study, we develop *storeds-bench* to help developers better understand the overhead they will encounter when using a certain data structure on PMEM. Specifically, *storeds-bench* is designed as a benchmark suite that leverages YCSB and has various commonly used storage data structures implemented using PMDK (persistent memory develop kit) under different persistent and consistency requirements.

CCS Concepts • **Information systems → Phase change memory**; • **Theory of computation** → *Data structures design and analysis.*

Keywords Persistent Memory, Data Structure, Benchmark

1 Introduction

Non-volatile memory (NVM) or persistent memory (PMEM) is a new kind of memory devices that provide near-DRAM data access latency and data persistence after power-off [6]. Among all the non-volatile memory solutions, Optane Persistent Memory (3D XPoint) from Intel is the first released product on the market [3]. Table 1 compares the specs of DRAM and Optane PMEM [7]. It is seen that Optane has a larger per-dimm size (high density) and lower per GB cost with relatively good bandwidth and latency. PMEM is considered as a promising complement of DRAM in the memory hierarchy.

PPoPP '20, February 22–26, 2020, San Diego, CA, USA
© 2020 Copyright held by the owner/author(s).
ACM ISBN 978-1-4503-6818-6/20/02…$15.00
https://doi.org/10.1145/3332466.3374509

Table 1. Intel Optane v.s. DRAM[1]

Device	Price	$/GB	Band. (GB/s)	Latency (ns)
DRAM-128GB	≈$2,400	18	≈80 [w/r]	≈120 [w/r]
Optane-128GB	≈$700	5	≈10 [w] ≈40 [r]	≈800 [w] ≈400 [r]
Optane-256GB	≈$2,400	10	-	-
Optane-512GB	≈$7,000	15	-	-

2 PMEM Programming Overheads

Although PMEM seems promising for efficiently storing data, programming it does need extra care to guarantee data consistency. This complicates the performance overheads more than just the hardware difference.

First, although PMEM itself is non-volatile, the caches and registers that CPUs utilize to access PMEM are still volatile. So, just calling `store` to persistent memory address does not guarantee the data persistence, as the data might still in the cache and will be lost after the system reboot. It is then necessary to call `CLFLUSHOPT` to force the cache line flush. This will introduce overheads for each persistent write. Also, modern CPUs reorder memory accesses. So, for two consecutive persistent memory writes, we may only have the later one persisted after failures, violating the order and potentially breaking data consistency. To avoid this, we have to enforce the order of multiple memory operations by calling memory fence, such as `CLWB` after flushing the cache, which also introduces overheads [1].

Second, the atomic write unit for modern CPUs is small (e.g., 8-byte for Intel Cascade Lake CPUs), so even with cache line flushing and memory fencing, writing a large chunk of memory to PMEM may still result in partially persisted data if there are failures in the middle. To guarantee data consistency, we will have to deploy mechanisms such as logging (undo log and redo log) or copy-on-write (CoW) to protect any write that is larger than 8 bytes. This complicates the programming itself and introduces significant overheads.

These extra programming overheads and the asymmetric physical performance of persistent memory on reads and writes may change the performance characteristics of data

[1] *The bandwidth and latency numbers are collected for the Optane-128GB model [7]. The actual price may vary.*

structures on persistent memory. Understanding the actual overheads model of PMEM data structures under different workloads then becomes critical for developers. In this study, we proposed *storeds-bench* to help tackle this challenge.

3 Design and Implementation

We first category the PMEM overheads into two parts:

- *Persistence Overhead*, which denotes the overheads caused by persisting data items into PMEM. This includes both its hardware overhead as well as the extra operations of flushing cache lines and fencing memory operations. Such overheads are necessary as long as developers expect the data (no matter it is small or large) to be persistent after failures.
- *Transaction Overhead*, which denotes the overheads whenever we need to persist a set of memory operations together. In this case, developers have to use transactions to protect these operations. Otherwise, it is possible to leave partial results persisted.

To exam these overheads, in *storeds-bench*, we implemented the storage data structures in four different versions: DRAM, PMEM-Volatile, PMDK-Persist, PMDK-Trans for comparison. Here, DRAM version is the traditional data structure implementation in DRAM, without any persistence. PMEM-Volatile is the usage of persistent memory device as a volatile memory with the support of PMDK `libvmem` [4] library. Both PMDK-Persist and PMDK-Trans use the PMDK `libpmemobj` library to implement [5]. PMDK-Persist leveraged `_persist` APIs to guarantee single write persistence. PMDK-Trans further leveraged PMDK transaction APIs to guarantee multiple memory accesses consistency, which is necessary for operations like allocating a new entry and linking it into a list.

We leveraged the YCSB workload generator to create different workloads for the evaluations [2]. We used 8 Byte integer key, 128 Byte string value, and, Zipfian type request distribution for most of the tests.

4 Initial Results and Analysis

We reported our initial results in Fig. 1. In this test, we evaluated the performance of four different versions (DRAM, PMEM-Volatile, PMEM-Persist, PMEM-Trans) of four example data structures, i.e., `hashtable`, `array`, `skiplist`, and `b-tree`, on five different YCSB workloads. Note that, all these data structures are implemented following basic strategies as our goal is to exam how data structures are affected by PMEM instead of finding the best implementation.

From these results, we have several initial observations. First, PMEM persistence and transaction do have negative impacts on write performance. For instance, `b-tree` PMEM-Trans version performs 29x worse than its DRAM counterpart on write-dominated (i.e., 0(R)/100(W)) workload. Second, PMEM devices have much less overhead on read-dominate (i.e., 100(R)/0(W)) workload. In fact, PMEM-Volatile `b-tree`

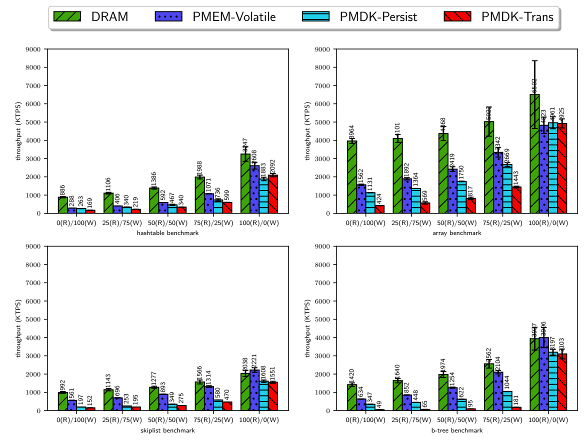

Figure 1. *storeds-bench* results. *y-axis shows the throughput (K operations per second); x-axis shows different workloads.*

and `skiplist` has a similar performance as its DRAM version. Given PMEM is physically slower than DRAM, we believe such a result should come from the cache behaviors. It is also noticeable that PMEM has different impacts on different data structures on a read-dominated workload. For instance, `b-tree` and `hashtable` have almost the same DRAM performance, while `hashtable` performs much worse than `b-tree` on PMEM. Last but not the least observation is that, in our current implementation, a data structure that performs better in DRAM may perform worse in PMEM. For example, for write-dominated workload (i.e., 0(R)/100(W)), `b-tree` performs better than `skiplist` on DRAM (1420 vs. 992), but performs worse on PMEM-Trans (49 vs. 152). Such a result shows the necessity of having *storeds-bench* to benchmark the overheads of data structures before making the design decisions, and exemplify that data structures performance on PMEM could diverge from its traditional DRAM version.

References

[1] Shimin Chen and Qin Jin. 2015. Persistent b+-trees in non-volatile main memory. *Proceedings of the VLDB Endowment* 8, 7 (2015), 786–797.
[2] Brian F Cooper, Adam Silberstein, Erwin Tam, Raghu Ramakrishnan, and Russell Sears. 2010. Benchmarking cloud serving systems with YCSB. In *Proceedings of the 1st ACM symposium on Cloud computing*. ACM, 143–154.
[3] Optane. 2019. Intel Optane Persistent Memory. https://www.intel.com/content/www/us/en/products/docs/memory-storage/optane-persistent-memory/optane-dc-persistent-memory-brief.html. Accessed: 2019-11.
[4] pmem.io Persistent. 2019. libvmem and libvmmalloc: malloc-like volatile allocations. https://github.com/pmem/vmem. Accessed: 2019-11.
[5] pmem.io Persistent. 2019. Persistent Memory Programming. https://pmem.io. Accessed: 2019-11.
[6] Kosuke Suzuki and Steven Swanson. 2015. A survey of trends in non-volatile memory technologies: 2000-2014. In *2015 IEEE International Memory Workshop (IMW)*. IEEE, 1–4.
[7] Alexander van Renen, Lukas Vogel, Viktor Leis, Thomas Neumann, and Alfons Kemper. 2019. Persistent Memory I/O Primitives. *arXiv preprint arXiv:1904.01614*.