Rethinking the Performance/Cost of Persistent Memory and SSDs

Kaisong Huang[1], Darien Imai[1], Tianzheng Wang[1], Dong Xie[2]
[1] Simon Fraser University, Canada [2] The Pennsylvania State University, USA

1 THE STORAGE JUNGLE

For decades, the storage hierarchy consisted of layers with distinct performance characteristics and costs: a higher level (in particular, memory) is assumed to be strictly faster, less capacious, volatile, and more expensive than a lower-level layer (e.g., SSDs and HDDs).

This *good ol'* storage hierarchy, however, is becoming a jungle: On the one hand, persistent memory (PM) breaks the boundary between volatile and non-volatile storage with persistence on the memory bus. On the other hand, modern SSDs' high bandwidth directly rivals PM, breaking the strict hierarchy from the performance perspective. For example, the Intel Optane P4800X/P5800X SSDs—using the same material as the Optane DCPMM's but in NVMe interfaces—offer up to 2.6–7.4GB/s of bandwidth and $< 6\mu$s latency [4]. Yet a server fully populated with the 100-series Optane DCPMM can offer up to ~7GB/s (random) to 40GB/s (sequential) bandwidth for reads, and ~4GB/s to 10GB/s peak bandwidth for writes [5]. This naturally leads to a simple, motivating question: *Could a well-tuned SSD-based data structure (e.g., index) match or outperform a well-tuned PM-tailored data structure under certain workloads?* In contrast to SSD-based systems, the CPU cost can be non-trivial as PM requires relatively high-end CPU support, which leads to the other question: *How does the "real" cost of a PM-based system stack up and compare to that of an SSD-based system?*

These advances and questions signal the need to revisit the performance/cost of persistent data structures. We take B+-trees and hash tables for an initial inquiry. Our goals are to (1) understand the relative merits of indexing on PM and SSD, (2) reason about the cost of PM- and SSD-based systems, and (3) highlight the implications of the storage jungle on future persistent indexes.

2 HOW MUCH DOES IT COST, REALLY?

When considering the cost of a storage system, we need to take into account three factors: memory (DRAM), CPU and the actual storage devices [7]. For fair comparison, we use a single server and vary the storage (SSD/PM) components using common and recommended setups. The server is equipped with a 20-core/40-thread Intel Xeon Gold 6242R CPU. It supports NVMe over PCIe Gen3 and 100-series DCPMMs. Each of the CPU's six memory channels is populated with one 32GB DRAM DIMM for a total of 192GB, leaving one slot per channel for PM. As recommended [3], we populate all channels with DRAM for maximum bandwidth. Both DRAM and DCPMM frequencies are fixed to 2666 MT/s for all the configurations.

We compare various configurations using different numbers of Optane DCPMMs and Optane SSD. As summarized in Table 1 (more details in our full paper [2]), we make two key observations: (1) A fully populated setup usually achieves the best cost/capacity for PM-based systems thanks to interleaving. (2) Although Optane DCPMM's raw cost is only 1.6× the cost of Optane SSD, its gross cost including CPU and DRAM can be over 4–5× the cost of Optane SSD; this is largely a result of strict memory population rules, whereas SSDs do not have similar restrictions.

[*] Full paper in CIDR 2022 [2]: http://cidrdb.org/cidr2022/papers/p64-huang.pdf.

Table 1: Storage cost (USD) of three configurations, using one/six (PM1/PM6) DCPMMs and one P4800X (P4800X1)

Component	PM1	PM6	P4800X1
CPU (Xeon Gold 6242R)	$2,517	$2,517	$2,517
DRAM (6×32GB)	$1,157.94	$1,157.94	$1,157.94
DCPMM (n×128GB)	$546.75	$3,280.50	N/A
P4800X SSD (m×375GB)	N/A	N/A	$999
Total	$4,221.69	$6,955.44	$4,673.94
Per GB (storage-only)	$4.27	$4.27	$2.66
Per GB without CPU	$13.32	$5.78	$5.75
Per GB with full CPU	$32.98	$9.06	$12.46
Per GB with 1 thread	$13.81	$5.86	$5.92
Per GB with 5 threads	$15.78	$6.19	N/A
Per GB with 10 threads	$18.23	$6.60	N/A

3 PUTTING COSTS IN THE INDEX CONTEXT

Using the configurations from Section 2, we compare the throughput and performance per dollar trends of PM and SSD based indexes.

3.1 Experimental Setup

Raw Bandwidth. Before testing indexes, we calibrate our expectations by testing raw read/write bandwidth of different configurations using the popular `fio` tool. For PM, we use `fio`'s `libpmem` backend which uses DAX/mmap to bypass the file system and expose byte-addressability. For SSD we use the `SYNC` and `io_uring` backends which represent the traditional `O_SYNC` and new asynchronous I/O, respectively.

Persistent Indexes. We use FPTree [9] and BzTree [1] to respectively represent PM trees that leverage DRAM to store inner nodes and those that only use PM. For PM-based hash tables, we use Dash [8]. For all indexes we use the settings recommended by their original papers or more recent evaluations [6]. We implemented a simple SSD B+Tree and a hash table that access data through a buffer pool atop `P4800X1`. For simplicity, we support concurrency with thread-partitioned key space; for fairness we use the same strategy for PM indexes. All I/Os are done using POSIX `pread`/`pwrite` with `O_DIRECT` to avoid OS caching. As we will see, even sub-optimal implementations can give very competitive performance/cost compared to very well-tuned PM-based indexes.

3.2 Evaluation

Raw Storage Performance. At first glance, PM setups outperform `P4800X1` in most cases (Figure 1), showing PM's advantage of being on the memory bus. However, a fair comparison to `P4800X1` would be `PM1` which consists of a single DCPMM without any interleaving,

whereas PM4 and PM6 leverage interleaving to gain performance advantages. P4800X1 using io_uring (P4800X-IU) exhibits sustained, stable performance regardless of access patterns (we only show sequential in Figure 1) or the number of threads thanks to the asynchronous programming model. This corroborates with prior work that P4800X needs very few threads to be saturated. Since newer SSDs typically utilize PCIe Gen4 which is not supported by our server, we plot the advertised bandwidth (dashed lines) in Figure 1 for reference. P5800X is expected to outperform PM4 under most write workloads and can match PM6 at high concurrency where PM6 does not scale well; notably, P5800X is priced similarly to P4800X. These results verify the advantage of SSD's asynchronous programming model over PM's synchronous programming model.

Index Performance/Cost. To obtain the performance/cost ratio R under a particular workload and index, we divide the measured throughput P by the storage system cost, including storage device cost $S, DRAM cost $D, and CPU cost $E:

$$R = \frac{P}{\$S + \$D + \$E} = \frac{P}{\$S + \$D + (W * U) * (\$C * \frac{1}{T})} \quad (1)$$

In Equation 1, $E is deduced from the number of worker threads W, the average CPU utilization U (out of 100%, representing the on-CPU time for data movement) during the run, the CPU's price $C and the number of total hardware threads T. Here, U is needed because for P4800X1 the system is often I/O bound without fully using the CPU; in many real applications such off-CPU time can overlap with computation. Thus, $U \times W$ yields the "effective" number of threads needed by a workload. $\$C/T$ gives the per-thread cost.

As shown in Figure 2, we calculate performance/cost ratios and make five observations: (1) A single P4800X SSD exhibits similar cost per performance to one DCPMM. (2) Interleaving is necessary for PM to perform well at high concurrency; this in turn requires the server be equipped with enough (more) CPU cores to support application logic. (3) Among the PM setups, PM4 can be more cost-effective than PM6. (4) PM-based indexes should utilize the necessary amounts of DRAM to increase performance and to amortize cost. (5) For memory-resident workloads, an SSD-based system can be fast at a low cost; for partial-memory workloads, it exhibits great potential with new asynchronous I/O.

4 IMPLICATIONS AND OUTLOOK

We now distill several high-level implications on index/system designs in the storage jungle:

Don't forget about SSDs yet! The DRAM-SSD hierarchy is still very cost-effective and should be considered first before using PM. This is especially true for memory-resident workloads where the benefits of adding DRAM far outweighs its cost. However, in case the application does require PM-level latency, PM may be a better (and likely the only) choice.

The PM stack can be equally or more expensive than the storage stack. The latter is often blanketly blamed as having high software overhead, but PM's synchronous/memory nature and rigid population rules require higher TCA with more cycles of high-end CPUs for data movement. This also implies that in future PM systems it is desirable to employ more CPU cores to satisfy the need of running application logic and moving data around. In contrast, the storage stack is increasingly lightweight and less CPU intensive.

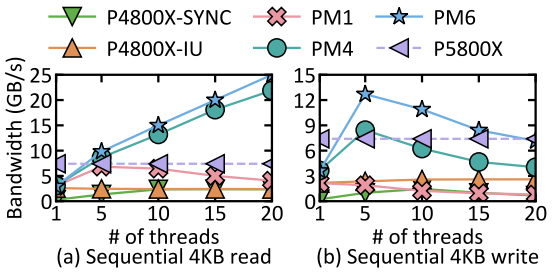

Figure 1: Raw sequential read/write bandwidth.

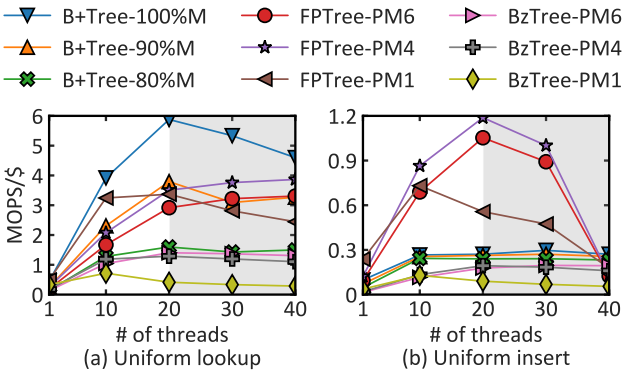

Figure 2: Performance/cost ratios of range indexes..

Co-design of hardware configuration and data structure is (more) necessary for PM systems. Otherwise, overprovisioning can considerably drive up TCA.

Outlook. Although both PM and SSDs are fast evolving, we believe that for PM to be truly cost-effective, it is necessary for its price to further drop. Since caching stores remain very cost-effective, an important direction is to explore how the cost-effectiveness equation changes when PM is used as an extension to DRAM (e.g., in Optane DCPMM's Memory mode). In addition, new SSDs are bringing much more potential for building fast persistent data structures and systems with the aforementioned performance characteristics and new abstractions such as user-space I/O, ZNS and directives.

REFERENCES

[1] Joy Arulraj et al. 2018. BzTree: A high-performance latch-free range index for Non-Volatile memory. *PVLDB* 11, 5 (Jan. 2018), 553–565.
[2] Kaisong Huang, Darien Imai, Tianzheng Wang, and Dong Xie. 2022. SSDs Striking Back: The Storage Jungle and Its Implications on Persistent Indexes. In *12th Annual Conference on Innovative Data Systems Research, CIDR*. 9–12.
[3] Intel Corporation. 2020. Intel Optane Persistent Memory Start Up Guide.
[4] Intel Corporation. 2021. Optane SSD P5800X Series Product Brief.
[5] Joseph Izraelevitz et al. 2019. Basic Performance Measurements of the Intel Optane DC Persistent Memory Module. *CoRR* abs/1903.05714 (2019).
[6] Lucas Lersch, Xiangpeng Hao, Ismail Oukid, Tianzheng Wang, and Thomas Willhalm. 2019. Evaluating persistent memory range indexes. *PVLDB* 13, 4 (2019).
[7] David Lomet. 2018. Cost/Performance in Modern Data Stores: How Data Caching Systems Succeed. *DaMoN* (2018).
[8] Baotong Lu, Xiangpeng Hao, Tianzheng Wang, and Eric Lo. 2020. Dash: Scalable Hashing on Persistent Memory. *PVLDB* 13, 8 (April 2020), 1147–1161.
[9] Ismail Oukid, Johan Lasperas, Anisoara Nica, Thomas Willhalm, and Wolfgang Lehner. 2016. FPTree: A hybrid SCM-DRAM persistent and concurrent B-tree for Storage Class Memory. *SIGMOD* (2016).