SSDs Striking Back: The Storage Jungle and Its Implications on Persistent Indexes

Kaisong Huang
Simon Fraser University
kha85@sfu.ca

Darien Imai
Simon Fraser University
darieni@sfu.ca

Tianzheng Wang
Simon Fraser University
tzwang@sfu.ca

Dong Xie
The Pennsylvania State University
dongx@psu.edu

ABSTRACT

The recent exciting development of persistent memory (PM) has led to many new proposals that directly operate and persist indexes on the memory bus, potentially removing the need for the storage stack. However, next-generation SSDs are quickly catching up with performance that overlaps with PM, effectively turning the storage hierarchy into a storage *jungle*. It is unclear how future persistent indexes (and data structures in general) should be designed, and more importantly, how their performance/cost would change given PM's unconventional installation requirements compared to SSDs.

This paper takes a first step to revisit the overall system cost and performance characteristics of the storage jungle, in the context of persistent indexes. We do so by experimentally evaluating PM and SSD indexes under real-world hardware constraints. We find that although PM has its own set of advantages, traditional DRAM-SSD hierarchies continue to be more cost-effective, and there is much to be further unleashed. Through careful analysis, we distill a series of observations, implications, and outlook on future index designs to navigate through the storage jungle.

1 INTRODUCTION

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 (in particular, SSDs and HDDs). Balancing performance and cost, such hierarchy has led to the continued success of caching stores [21] that use volatile memory (DRAM) as a caching layer atop persistent storage (SSDs and HDDs), which many systems are based upon. This *good ol'* storage hierarchy, however, is undergoing disruptive changes with two trends led by persistent memory (PM) and ultra-fast SSDs.

Trend 1: Memory Meets Persistence. Scalable PM media such as 3D XPoint [11], PCM [34] and STT-RAM [10] attempt to approach DRAM speeds while providing byte-addressability, persistence, and SSD-level high capacity, yet at a lower cost than DRAM's. An actual PM product, as represented by Intel Optane DCPMM (based on 3D XPoint), however, exhibits a significant performance gap between DRAM[16]. For example, a server fully populated with the 100-series Optane DCPMM can offer up to 12TB of capacity, up to ~7GB/s (random) to 40GB/s (sequential) bandwidth for reads, and ~4GB/s to 10GB/s peak bandwidth for writes [16]. Therefore, they present a significant improvement of performance compared to traditional SATA SSDs and offer much lower and more stable latency (~300ns). The recently released 200-series DCPMM further promise ~25% higher bandwidth [14]. Proposals from the "PM camp" advocate single-level stores that directly operate and persist data on PM, enabling low-latency durable commits for applications that potentially need it; SSDs and HDDs are relegated to archival and backup purposes, or may even be omitted completely. A representative body of work is PM-tailored indexes [2, 20, 22, 28].

Trend 2: Storage Approaches (Persistent) Memory. Meanwhile, new generations of SSDs continue to achieve unprecedented high bandwidth and low latency from the adoption of new materials such as 3D XPoint, hardware design such as 3D NAND, and faster interfaces such as NVMe over PCIe Gen4. For example, the 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 [15]. The Samsung 980 PRO based on NAND flash can offer up to ~5–7GB/s bandwidth on a PCIe Gen4 ×4 link [30]. Motivated by recent advances in SSDs, better caching stores [17, 23] have also been proposed to approach in-memory performance. Driven by faster storage, I/O interfaces have also evolved to be much more lightweight with new programming models such as SPDK [32] and io_uring [7], further reducing overhead caused by the storage stack at the software level.

The Hierarchy is Becoming a Jungle. PM breaks the boundary between volatile and non-volatile storage with persistence on the memory bus. Modern SSDs' high bandwidth directly rivals PM, breaking the strict hierarchy from the performance perspective [36]. SSDs still exhibit higher latency than PM, and thus, would not provide the same low-latency durable commit using PM-based indexes. However, many applications (including a lot of OLTP use cases) are throughput-oriented and involve many layers (e.g., networking in a DBMS). As the latency gap between SSD and PM continues to shrink, PM's low-latency commit may not tremendously benefit them. 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?*

Conventional wisdom about storage system cost is also being challenged as PM and SSD assume different programming models: the former is synchronous with CPU `load` and `store` instructions, and the latter is asynchronous with DMA. PM-tailored data structures must dedicate more CPU cores (for “I/O”) to saturate the system. Conversely, SSD-based data structures may overlap computation with I/O, leaving more cores for “real” work; it has been shown that using SPDK, a single core is enough to saturate multiple SSDs [32]. In a PM-based system, CPU cycles can become a scarce resource that have to be timeshared between computation and data movement. In contrast to SSD-based systems, the CPU cost can be non-trivial as PM requires relatively high-end CPU support. This leads to another question: *How does the “real” cost of a PM-based system stack up and compare to that of an SSD-based system?*

Contributions. These recent advances and questions signal the need to revisit the performance/cost of persistent data structures. In this paper, we take persistent B-trees and hash tables for an initial inquiry. Our goals are to (1) understand the relative merits of index designs 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. To reach these goals, for the first time, we experimentally compare PM-tailored indexes [2, 22, 28] against SSD-based indexes that use a DRAM buffer pool atop a modern SSD. We consider several representative hardware configurations and analyze their costs under index workloads. We highlight several findings here and expand on details later:

- CPU cost is non-trivial and subverts the cost-effectiveness of a PM-based system. Although polling has started to become necessary for SSDs, it requires much fewer threads.
- A traditional DRAM-SSD hierarchy continues to be increasingly cost-effective; DRAM’s higher-than-PM price is offset by SSD’s even lower per-byte cost and demand on CPU cycles.
- PM prices need to drop for true cost-effectiveness. For memory-resident workloads (e.g., many OLTP workloads), caching stores are more cost-effective than PM, mainly due to PM’s population rules that overprovision DRAM and CPU.

Our focus is performance/purchasing cost (i.e., total cost of acquisition/TCA without considering operational costs). We use a server that is actually equipped with the devices under discussion. Competitive prices were obtained online as of this writing (August/September 2021) [6, 24–26]. Code and instructions for our experiments are available at https://github.com/sfu-dis/ssd-vs-pm.

2 THE MODERN STORAGE JUNGLE

As we briefly described in Section 1, the storage hierarchy is becoming a *jungle* that features devices with overlapping performance characteristics. Here we give more details on modern PM and SSDs being introduced to the storage hierarchy.

2.1 Persistent Memory and Optane DCPMM

“PM” can refer to a wide range of devices built with different materials [10, 11, 31, 34]. Their common vision is to overcome DRAM’s limitations in capacity and energy consumption. As a side effect, however, they are persistent on the memory bus. Currently, “PM” is practically synonymous with Intel Optane DC Persistent Memory Module (DCPMM) as it is the only PM that scales to a high capacity (128/256/512GB per DIMM). NVDIMMs [1] based on DRAM backed by flash and supercapacitors/batteries have long been available, but are limited by DRAM’s capacity. Thus, we target DCPMM.

Compared to DRAM or SSDs, adopting DCPMM is far from “plug-and-play.” PM exhibits lower random/write bandwidth than sequential/read bandwidth. PM programming inherits memory’s synchronous nature by using `load` and `store` instructions rather than storage primitives leveraging interrupts or polling with DMA. These can incur more CPU stalls and require new programming paradigms. There are also unconventional installation requirements [12]:

- DRAM must be present to “enable” PM on the memory bus (i.e., a system cannot run without DRAM);
- DRAM DIMMs and DCPMMs must be plugged into memory slots following certain population and interleaving rules;
- Relatively high-end processors (e.g., 2nd/3rd-generation Xeon Scalable processors) are required.

Optane DCPMMs can operate in Memory, App Direct, or Dual modes. Memory mode leverages the underlying material’s scalability to extend volatile memory capacity; the system’s DRAM is used as a cache managed by the CPU/memory controller transparent to software, and data is erased across power cycles. App Direct offers persistence and allows software to judiciously place data in DRAM and PM. The Dual mode allows partitioning DCPMM to be in both Memory and App Direct modes. All the modes can be used with or without hardware interleaving which can improve bandwidth under high concurrency. Since we aim to explore DCPMM’s impact on persistent indexes, we focus on the App Direct mode.

Many recent studies [19, 38] have noted DCPMM’s performance characteristics, but did not significantly consider its cost. We consider both and analyze the TCA for PM-based systems in Section 3.

2.2 Next-Generation Fast SSDs

Modern, fast SSDs are typically built using NAND flash memory and are attached to the PCIe bus using an NVMe interface. Flash exhibits different properties to disks or PM: it can be programmed by page, but modifications must be done in blocks which consist of multiple pages. Such properties, however, are hidden from software using a flash translation layer (FTL) which is part of the SSD and presents to software the block interface like that of HDDs. Flash SSDs are significantly faster than HDDs and often exhibit very high internal parallelism, so software must issue enough requests to saturate the drive (measured by queue depth). Flash SSDs also exhibit a high level of read/write asymmetry with reads and sequential accesses being faster in general. Internally, the FTL manages multiple flash chips and optimizes performance and lifetime. With the recent advances in 3D NAND, QLC, and PCIe Gen4, SSDs have started to achieve even higher density and performance with a lower price, blurring the boundary between PM and SSDs.

In addition to flash, 3D XPoint can also be used to build SSDs (e.g., Intel Optane DC P4800X/P5800X). They use the same material as Optane DCPMM’s, but interface with NVMe over PCIe without providing byte-addressability. Like flash SSDs, a block interface is presented to software. In contrast to flash SSDs and DCPMM, Optane DC SSDs exhibit much less asymmetry with almost the same read/write and sequential/random access speeds [36]. In this paper, we conduct experiments and analyze the cost per performance using the 100-series DCPMM and Optane DC P4800X SSD; both use the same generation of 3D XPoint.

Table 1: Cost (USD) of five server configurations with different numbers of DCPMMs and SSDs, where PM*n* represents a configuration with *n* DCPMMs, and P4800X*m* represents a configuration with *m* P4800X drives. Per GB (without CPU) highlights the additional cost of DRAM, compared to Per GB (storage only) which only considers the storage media (DCPMM or P4800X). The other per GB numbers take into consideration the amount of CPU cycles that may be required to saturate the storage component. `P4800X1` and `P4800X2` require only a single thread to saturate the drive(s) and exhibit the lowest costs (Per GB with 1 thread). DCPMM configurations usually require more threads which drive up the per GB cost (Per GB with 1/5/10 threads) by up to over 4× (`PM1` with ten threads vs. `P4800X2` with one thread).

Component	`PM1`	`PM4`	`PM6`	`P4800X1`	`P4800X2`
CPU (1×Intel Xeon Gold 6242R, 20-core, 40-hyperthread)	$2,517	$2,517	$2,517	$2,517	$2,517
DDR4 DRAM (6×32GB)	$1,157.94	$1,157.94	$1,157.94	$1,157.94	$1,157.94
Optane DCPMM (n×128GB)	$546.75	$2,187.00	$3,280.50	N/A	N/A
Optane SSD P4800X (m×375GB)	N/A	N/A	N/A	$999	$1,998
Total	$4,221.69	$5,861.94	$6,955.44	$4,673.94	$5,672.94
Per GB (storage-only)	$4.27	$4.27	$4.27	$2.66	$2.66
Per GB without CPU	$13.32	$6.53	$5.78	$5.75	$4.21
Per GB with full CPU	$32.98	$11.45	$9.06	$12.46	$7.56
Per GB with 1 thread	$13.81	$6.66	$5.86	$5.92	$4.29
Per GB with 5 threads	$15.78	$7.15	$6.19	N/A	N/A
Per GB with 10 threads	$18.23	$7.76	$6.60	N/A	N/A

With recent development in NVMe, new storage interfaces and abstractions (e.g., Open-Channle [4, 29], ZNS [3] and FTL directives) are emerging to overcome the limitations of the block POSIX I/O interface for SSDs. They present new opportunities to further optimizing the storage stack [18]. For simplicity we use the traditional POSIX I/O interfaces for experiments on SSDs.

2.3 From a Hierarchy to a Jungle

As we briefly mentioned in Section 1, the clear delineation of layers has faded in terms of both performance and cost. As of this writing, a single 128GB Optane DCPMM ($546, or ~$4.27 per GB) is more expensive in unit price than a 400GB Intel Optane P5800X SSD ($1,200, or $3 per GB), while offering lower read and write throughput (6.6GB/s vs. 7.2GB/s and 2.3GB/s vs. 6.1GB/s). The disparity grows with density: a 256GB/512GB DCPMM is ~3.8×/~11.2× the price of a 128GB DCPMM [11]. Although Optane SSDs exhibit higher latency (3µs vs. 300ns), the gap is often too small to make a significant difference in end-to-end performance for many applications [9, 37]. This means users could pay less to achieve the same or higher performance using SSDs. It is even true for flash SSDs: for example, Samsung 980 PRO offers up to 7GB/s read and 5GB/s write bandwidth with a much lower price than the P5800X. It can achieve up to 1M IOPS for certain random I/O workloads, which is close to what P5800X can offer.

With all these facts, the different storage media are effectively forming a *jungle* instead of a hierarchy. Hence, to achieve a lower cost for sustaining a given workload, more careful design choices have to be made based on various workload and QoS requirements.

Our analysis so far focused on the cost and performance of individual storage media. We next take whole-system configurations to analyze their total acquisition cost and performance.

3 HOW MUCH DOES IT COST, REALLY?

Now we consider the acquisition cost of PM- and SSD-based storage systems; the next section further considers performance.

3.1 Hardware Configurations

When considering the cost of a storage system, we need to take into account three factors: memory (DRAM), CPU and the actual storage devices [21]. 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 [12], we populate all channels with DRAM for maximum bandwidth. Both DRAM and DCPMM frequencies are fixed to 2666 MT/s for all the configurations.[1]

PM-based Systems. We analyze three vendor-recommended[2] PM setups [12] listed in Table 1, denoted as `PM`*n* where *n* is the number of DCPMMs. `PM1` and `PM4` respectively use one and four DCPMMs, and `PM6` uses six DCPMMs, representing a fully populated system. Except for `PM1`, only symmetric population of 100-series DCPMMs across sockets are supported [12], i.e., the DRAM/DCPMM population should be the same across all sockets for multi-socket machines. One also cannot mix capacities or generations of DCPMMs in a single server. For applications that prefer to use more than one CPU, users then may be forced to overprovision DRAM and DCPMM to use PM as the main home of data. In contrast, SSD-based systems do not have such requirements. For the purpose of our comparison, we use one CPU as over-provisioning significantly drives up the cost of PM-based systems.

SSD-based Systems. `P4800X1`/`P4800X2` in Table 1 represents a classic DRAM-SSD hierarchy that uses one/two 375GB Optane

[1]The CPU and DRAM support up to 2933 and 3200 MT/s respectively, but have to be clocked down to 2666 MT/s for 100-series DCPMM. We expect a pure DRAM-SSD system without any DCPMM to perform better than what we report here.
[2]Although alternatives exist, in practice and based on our experience working with vendors, most users opt for these recommended setups which are fully vendor-tested and supported. So we focus on vendor-recommended setups in this paper.

P4800X (PCIe Gen3) SSD(s) without any PM. Unlike DCPMM, SSD-based systems do not impose strict memory population rules. Differing capacities or generations of SSDs which potentially use different interfaces can be installed in the same system, and various storage schemes such as RAID can be used to achieve higher availability, performance and reliability. Capacity-wise, P4800X2 (750GB) and PM6 (768GB) would be fair comparisons. Performance-wise, 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 (RAID 0 SSD setups would then be fair comparison). Nonetheless, we compare all the listed setups to analylze their relative merits for completeness and explore the cost/capacity trends.

3.2 Cost/Capacity Analysis

With the configurations we evaluate five metrics listed in Table 1:

- Total: purchase price of all the storage-related components of the server (CPU, DRAM and DCPMM/SSD);
- Per GB (storage): the per GB price of the storage component (DCPMM or Optane SSD);
- Per GB w/o CPU: same as Per GB w/ CPU but with CPU cost excluded from Total, i.e., leaving DRAM and storage costs only;
- Per GB w Full CPU: Total divided by storage capacity;
- Per GB w/ T thread(s): same as Per GB w/ Full CPU but calculated using the cost of T threads (prorated using the CPU's price). For SSD setups we only consider $T = 1$ because a single thread can already saturate the SSDs [32], while for PM it requires 5-10 threads [38] (details later).

Now we explain the cost and findings using the above metrics.

Finding 1: PM is more expensive by just being "memory." PM systems' Total ranges from ~$4200 to ~$7000 which is a linear function of the number of DCPMMs used. Note that these PM configurations exhibit different total storage capacities: PM1 has 128GB, PM4 has 512GB and PM6 has 768GB, where as P4800X costs $4673.94 with a total capacity of 375GB. The Total of 375GB P4800X is in fact comparable to that of 128GB PM1 (adding another DCPMM would make it even closer). This is attributed to the raw storage device cost: although using the same underlying storage material (3D XPoint), being on the memory bus makes the 128GB Optane DCPMM $1.61 more expensive per GB than Optane SSD as shown by the Per GB (storage-only) row in Table 1. Moreover, we note that DCPMM prices do not grow linearly by its capacity as Section 2.3 mentioned. In contrast, the price for Optane SSD scales almost perfectly linearly [13]; the gap further grows significantly with 256/512GB DCPMM.

Finding 2: Necessary non-storage components (CPU and DRAM) can dominate the cost of PM-based systems. Comparing Per GB (storage) and Per GB w/o CPU shows the impact of DRAM: for all configurations it adds non-trivial additional costs. Among them, PM6 exhibits the lowest cost as it best amortizes the cost of all DRAM DIMMs with six DCPMMs. Note that P4800X1 and P4800X2 are placed at an unnecessarily disadvantaged position: in reality, the server does not have to be fully populated with six DRAM DIMMs to properly operate–although rarely done, using one DRAM DIMM would suffice. Despite the disadvantage, P4800X1 and P4800X2 still exhibit lower costs than any PM setup.

The impact of CPU cost is more pronounced compared to DRAM's. Ideally, we should consider the actual number of cores used to access PM and SSD as under pure read/write microbenchmarks using 4–5 threads is enough to saturate random write bandwidth, while read can usually scale up to about 10 threads [38]. However, typically PM data structures need to use all or most of the cores to reach peak performance because of complex read/write mixes [19]. We thus compare the unit prices in the following two cases: (1) Taking the whole CPU into account, unsurprisingly, Per GB w/ CPU grows from PM6's ~$9/GB to PM1's ~$33/GB, and P4800X1/P4800X2 costs $12.46/$7.56 per GB. The trend roughly follows that of DRAM's impact, as PM6 best amortizes the CPU's cost. (2) The Per GB w/ T thread(s) numbers show examples under budget using one, five and ten cores. We choose these settings because a single thread can already saturate SSDs [32], and the PM configurations saturate at ~5-10 threads depending on access pattern, so we give a range here. Again, PM1 exhibits the highest unit cost of ~$13–18$/GB, compared to PM6's ~$6/GB, and P4800X1's $5.92/GB. PM1 also performs the slowest as interleaving cannot be used with one DCPMM. For some workloads, however, we suspect the capacity provided by PM6 to be excessive. Comparing PM6 and P4800X2, the latter's cost remains a constant of $4.29/GB (assuming a RAID 0 setup which resembles PM6's interleaved access), while PM6's cost grows linearly with the number of threads employed.

Finding 3: Capacity can be a determining factor in choosing between PM and SSD. With 4–6 DCPMMs, the unit costs of PM4 and PM6 are very similar to that of P4800X. However, a caveat is that PM-based systems have a hard upper bound on capacity: the largest DCPMM is 512GB, and with six channels a server usually can be equipped with up to 12TB of DCPMM in a quad-socket system. Yet by using multiple SSDs, a server's total capacity can exceed well beyond 12TB with better price/capacity scalability. Recent work has shown the potential of using DCPMM as a drop-in replacement for SSDs with promising results [5], but SSDs may still be the necessary choice over DCPMM for applications that need high storage capacity.

Summary. From the cost/capacity perspective, we summarize two key observations: (1) A fully populated setup usually achieves the best cost/capacity for PM-based systems, sometimes even better than SSD's; this coincides with the fact that a fully populated system also offers the best performance 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. As we briefly touched upon earlier, these memory population rules also impact performance of PM-based systems, which we explore next.

4 PUTTING COSTS IN THE INDEX CONTEXT

Using the configurations from Section 3, we compare the throughput and performance per dollar trends of PM and SSD based indexes.

4.1 Experimental Setup

We performed experiments using Ubuntu 20.04 LTS with Linux kernel 5.8.0-59.66, libpmem 1.8, and fio 3.26. All index implementations and benchmarking drivers were compiled using GCC 11.2

with the highest optimization level.[3] The CPU frequency governor was set to performance and PCIe ASPM was disabled in BIOS to avoid unstable performance caused by power management.

Raw Bandwidth. Before testing indexes, we calibrate our expectations by testing raw read/write bandwidth of different configurations using the popular `fio` tool. We test four access patterns (4KB sequential/random read/write) over a 500MB file. 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. Although prior work has done similar experiments [16], we vary PM population setups and compare PM with SSD performance under different I/O backends to highlight the impact of I/O programming models.

Persistent Indexes. We test three PM-tailored tree and hash table designs under the PM*n* configurations. We use FPTree[4] [28] and BzTree [2] 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 [22], a state-of-the-art design tailored for DCPMM. Specifically, we use its extendible hashing variant with an reclamation epoch period of 1024 accesses.

For all indexes we use the settings recommended by their original papers or more recent evaluations [19]. FPTree uses 128-record inner nodes and 64-record leaf nodes. For BzTree, we use a leaf node size of 1KB, which was found to have higher throughput than using 4KB nodes. FPTree and Dash use a pre-allocated 4GB PM pool for read tests and a 32GB PM pool for write tests. BzTree used a 32GB PM pool for both read and write tests.

We implemented a simple SSD B+Tree and a hash table that access data through a buffer pool atop `P4800X1`. The SSD-based hash table is static and uses closed-addressing; we reserve enough space for insert workloads. 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 the synchronous POSIX `pread`/`pwrite` with `O_DIRECT` to avoid OS caching. Note that our SSD-based index implementations are in fact *not* well-tuned for modern SSDs, nor follow state-of-the-art design principles [17, 33]. New I/O libraries such as SPDK have the potential of achieving much higher performance than our implementations [32]; for example, PA-Tree [33] can saturate an NVMe SSD with a single thread. Thus, using synchronous I/O puts our SSD-based indexes at a disadvantage compared to PM-based indexes. As we will see, however, even sub-optimal implementations can give very competitive performance/cost compared to very well-tuned PM-based indexes.

Index Workloads. We run the same set of workloads on all indexes by preloading the index with 100 million records (8-byte keys and 8-byte values), and then executing point lookup or insert operations. When performing lookups, the keys are drawn from a uniform or a skewed (0.99 skew factor) zipfian distribution; skew factors 0.2–0.6 showed similar trends to the results under the uniform distributions, so we omit them here.

[3]Code and instructions available at https://github.com/sfu-dis/ssd-vs-pm.

[4]FPTree is not the latest PM tree, but is one of the top performers [19]. We did not succeed in running all tests with the latest LB+-Tree [20] due to key overflow and isolation issues that impact performance (https://github.com/schencoding/lbtree). We hope to explore further but do not expect our conclusions to change fundamentally.

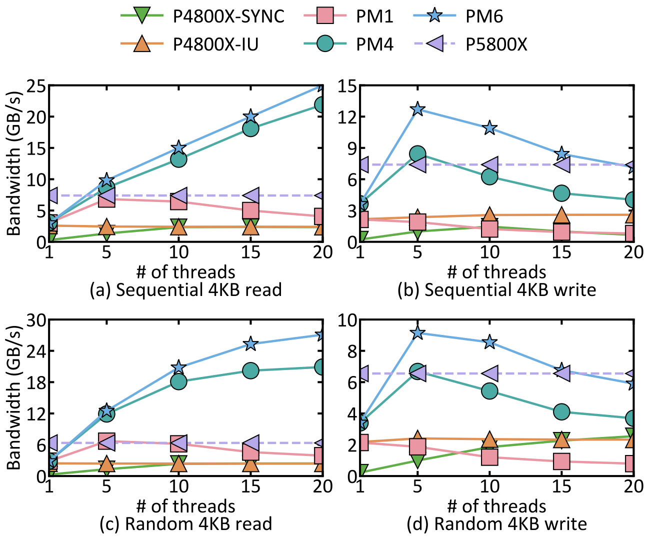

Figure 1: Raw sequential/random read/write bandwidth of six configurations. SSD's asynchronous programming interface shows advantage over PM's synchronous programming paradigm by requiring fewer threads to saturate, leaving more cycles for running useful application logic.

4.2 Raw Storage Performance

In terms of raw bandwidth, at first glance PM setups outperform `P4800X1` in most cases (Figure 1), showing PM's advantage of being on the memory bus. However, as Section 3.1 described, `PM4` and `PM6` are in fact not fair comparisons to `P4800X1`. Rather, `P4800X1` should be compared to `PM1`, which however not only costs the most per GB (Section 3), but also scales poorly in all cases, whereas `P4800X1` using io_uring (`P4800X-IU`) exhibits sustained, stable performance regardless of access patterns or the number of threads thanks to the asynchronous programming model. This corroborates with our earlier description and prior work that `P4800X` needs very few threads to be saturated. Since newer SSDs (e.g., Optane P5800X and Samsung 980 PRO) 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. We expect future SSDs to maintain such advantage.

4.3 Index Performance/Cost

We base performance/cost calculations on the `Total` row in Table 1 but only consider the actual CPU cycles used under each workload. This approach models the cost to access an index (out of other tasks that use the remaining CPU cycles) in an on-premise deployments where a server is purchased for providing a service.[5] To obtain the

[5]Equation 1 uses the whole storage component's cost; an alternative is to count only the storage space actually used, similar to what cloud vendors offer. We obtained similar trends under both methods, and so omit the results here.

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.[6] It is 100% for PM indexes because of PM's synchronous access nature, and ranged from ∼10 to 100% (when all the data is buffered) for P4800X1 as SSD I/O is blocking in nature. Thus, $U \times W$ yields the "effective" number of threads needed by a workload. Then, with the entire CPU's cost \$C, $\$C/T$ gives the per-thread cost. Next, we calculate performance/cost ratios based on the performance numbers obtained from our experiments.

Tree Indexes. For SSD-based B+trees we vary the buffer pool size between 80–100% of data size (denoted as B+Tree-x%M where x represents the percentage). B+trees can have a good memory hit ratio typically above 70%, since most of the inner nodes can be cached. In our case, 80% is a sweetspot, below which the throughput is bound by SSD bandwidth. Figure 2 shows the results; the gray areas indicate results obtained when hyperthreads are used. As shown by Figures 2(a–b), FPTree under PM4 and PM6 approaches the performance of the pure in-memory SSD-based B+Tree, because FPTree stores all inner nodes in DRAM (its performance collapses at 40 threads due to high HTM abort rates). BzTree performs much worse as it does not use any DRAM to store tree nodes. In Figures 2(d–e), caching stores (P4800X1) exhibit the best performance/cost ratio for lookup operations for memory-resident lookup workloads.

Once the workload is not memory-resident, at 90% cache ratio, the SSD-based B+Tree performs similarly to FPTree under PM1 for lookups under uniform distribution. When data access is skewed, it outperforms PM1. Notably, in most cases B+Tree-80%M gives the best performance/cost ratio although the SSD is fully saturated; it still performs better than BzTree with lower performance/cost ratio in Figure 2(e). Although we have not conducted experiments with the latest Optane P5800X SSD, based on these results, we expect that a smaller buffer pool is sufficient to get the same performance.

In Figure 2(f), FPTree's insert performance is much better with multiple DCPMMs and higher concurrency, thanks to its increased bandwidth from interleaving and lower latency. For inserts, a larger buffer pool does not necessarily improve performance. But in all cases, B+Tree-80%M gives much higher performance/cost ratios than the fully-PM BzTree, signaling the importance of using DRAM in PM indexes; Also, for FPTree, PM4 and PM6 give similar performance/cost, as PM4 already provides enough bandwidth.

Hash Tables. Figures 3(a-b) show that Dash scales the best for lookups. In contrast, the SSD-based hash table can hardly keep up with Dash for partial-memory workloads. However, inserts in Figure 3(c) exhibit a pattern that is quite different from that in Figure 2(c), with in-memory Hashtable-100%M gradually matching

[6]Database systems are among the classic examples of such applications, e.g., MySQL InnoDB uses Linux's asynchronous IO subsystem to schedule storage accesses [27].

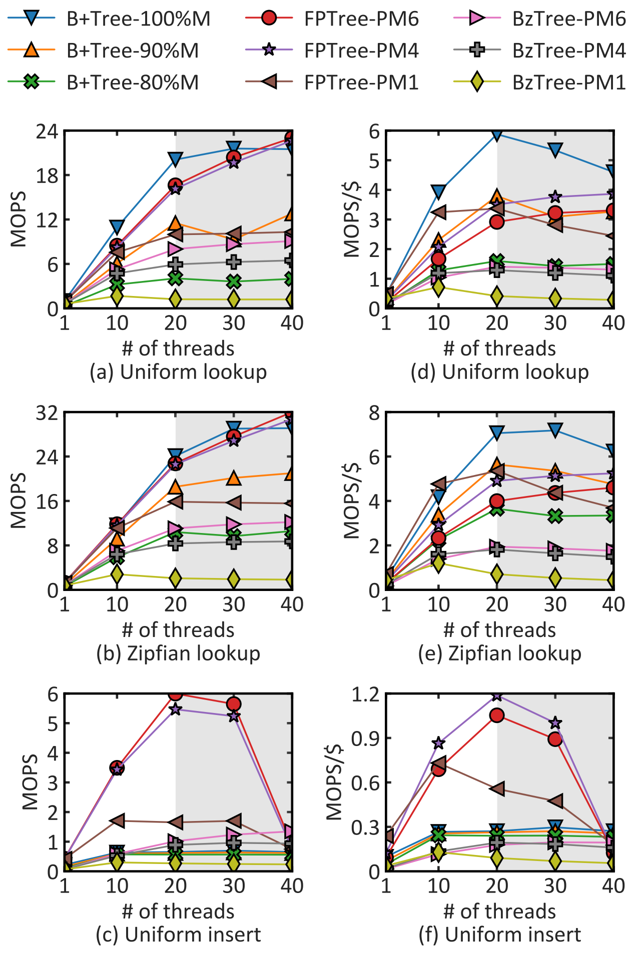

Figure 2: Throughput (million operations per second) (a–c) and performance/cost ratios (d–f) of range indexes.

and outperforming Dash. In terms of performance/cost ratios, Figures 3(d–e) again showed that memory-resident workloads exhibit the best cost-effectiveness, but for partial memory cases PM setups are more cost-effective. For inserts, however, P4800X1 showed similar and with hyperthreads even up to 2× better performance/cost ratio than PM6. Overall, these results show that PM-based hash tables should continue to optimize for insert performance to justify their costs, whereas SSD-based hash tables should further improve on probing performance.

Summary. We make five observations on performance per dollar: (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. It may not be always the best choice to fully populate a PM-based system, and the remaining DIMM slots could be used to populate more DRAM for better performance. (4) PM-based indexes should utilize the necessary amounts of DRAM to increase performance and to amortize cost; this coincides with findings reported by prior

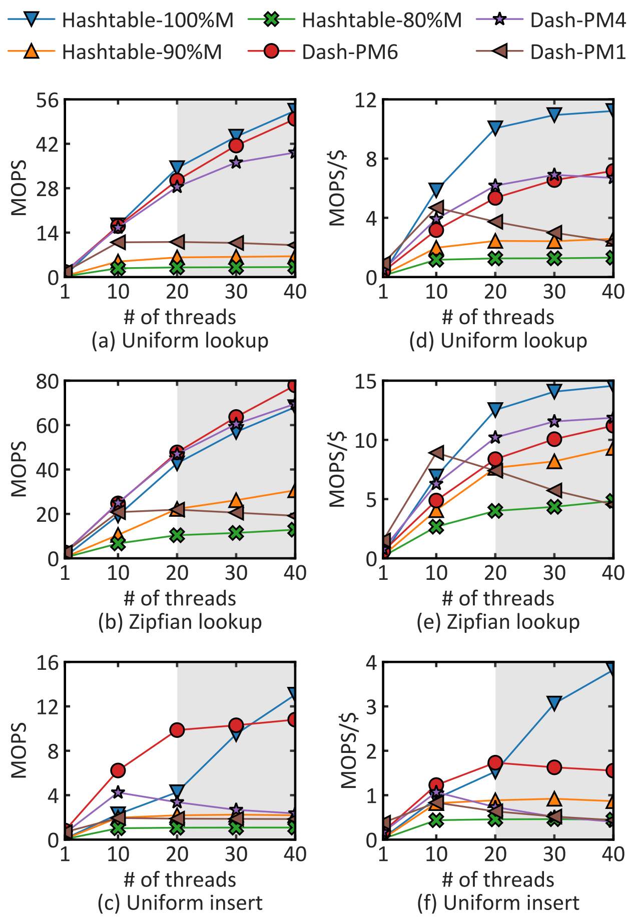

Figure 3: Throughput (million operations per second) (a–c) and performance/cost ratios (d–f) of hash tables.

work [19]. (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 and improving raw performance.

5 RELATED WORK

Our work is closely related to recent work on the performance/-cost of data systems, persistent data tructures and caching systems. Recent work has highlighted the potential of NVMe SSD arrays [9] and systems such as LeanStore [17] and Umbra [23] have demonstrated the feasibility of building caching systems that approach in-memory speed. PA-Tree [33] leverages modern asynchronous SPDK libraries to fully leverage modern NVMe SSDs with very low CPU resources. Lomet [21] discussed the cost/performance of DRAM-SSD systems and reasoned about the merits of in-memory vs. storage-centric designs. Programmable SSDs have also been shown to reduce cost/performance [8]. Meanwhile, PM-tailored data structures [20, 22, 28] use various techniques to overcome PM's slower-than-DRAM speed. Many designs store part of the index in DRAM for performance at the cost of instant recovery. In light of the storage jungle, Wu et al. explored the performance characteristics of Optane SSD [35] and introduced non-hierarchical caching [36] that utilizes Optane SSDs, DCPMMs, DRAM, and NAND flash SSDs. Our work highlights the performance/cost of the storage jungle in the context of persistent indexes. Exploring how performance/cost evolves in end-to-end systems is interesting future work.

6 IMPLICATIONS AND OUTLOOK

Despite the exciting development of PM, it is not a panacea, as modern SSDs are directly rivaling PM in both performance and cost aspects—even when our SSD-based indexes are neither the best ones known nor fully optimized.

In addition to the detailed observations made at the end of Sections 3 and 4, we distill several high-level implications and give outlook on future 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. As well, programming PM requires new skills with a steep learning curve, adding more implicit costs beyond TCA when compared to SSDs.

Co-design of hardware configuration and data structure is (more) necessary for PM systems. Otherwise, overprovisioning can considerably drive up TCA. For example, FPTree does not require a fully populated server to achieve peak performance and be cost-effective, but another data structure may require a very different configuration. Such co-design may limit the system's ability to evolve for different workloads, more so than SSD-based systems.

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 compared to pure-PM systems, 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 [3] and directives. From a practitioner's perspective, designing indexing structures that fully leverage the potential of modern SSDs is another promising direction. Our main focus has been simple index microbenchmarks. It would be interesting to explore the performance and cost of running more complex workloads (e.g., TPC-C, TPC-H) in an end-to-end system that leverage next-generation SSDs and/or PM. Finally, as we noted earlier, there are multiple candidate materials that can build PM, with some (e.g., STT-RAM [10]) even having the potential of approaching SRAM performance. It remains to be seen whether they can truly scale to high capacity with a low cost, which, again, would require a revisit of the storage *jungle*.

ACKNOWLEDGMENTS

We thank the anonymous reviewers for their constructive comments. We are tremendously grateful to Ryan Stutsman for his valuable discussions and comments on an earlier draft of this paper. Ge Shi, George He and Baotong Lu shared their valuable insights on PM-based indexes and PM programming. This work is partially supported by an NSERC Discovery Grant, Canada Foundation for Innovation John R. Evans Leaders Fund and the B.C. Knowledge Development Fund.

REFERENCES

[1] AgigaTech. 2017. Non-Volatile RAM. Retrieved August 17, 2021 from http://www.agigatech.com/nvram.php.

[2] Joy Arulraj, Justin Levandoski, Umar Farooq Minhas, and Per-Ake Larson. 2018. BzTree: A High-Performance Latch-Free Range Index for Non-Volatile Memory. *PVLDB* 11, 5 (2018), 553–565.

[3] Matias Bjørling. 2019. From Open-Channel SSDs to Zoned Namespaces. *Linux Storage and Filesystems Conference* (Feb. 2019).

[4] Matias Bjørling, Javier González, and Philippe Bonnet. 2017. LightNVM: The Linux Open-Channel SSD Subsystem. In *Proceedings of the 15th USENIX Conference on File and Storage Technologies (FAST'17)*. 359–373.

[5] Maximilian Böther, Otto Kißig, Lawrence Benson, and Tilmann Rabl. 2021. Drop It In Like It's Hot: An Analysis of Persistent Memory as a Drop-in Replacement for NVMe SSDs. In *Proceedings of the 17th International Workshop on Data Management on New Hardware (DaMoN 2021)*. Article 7, 8 pages.

[6] CDW. 2021. Intel Optane DC Persistent - DDR-T - 128 GB - DIMM 288-pin - 4 Pack. Retrieved August 17, 2021 from https://www.cdw.com/product/intel-optane-dc-persistent-ddr-t-128-gb-dimm-288-pin-4-pack/5749247.

[7] Jonathan Corbet. 2019. Ringing in a New Asynchronous I/O API. Retrieved August 17, 2021 from https://lwn.net/Articles/776703.

[8] Jaeyoung Do, Ivan Luiz Picoli, David Lomet, and Philippe Bonnet. 2021. Better Database Cost/Performance via Batched I/O on Programmable SSD. *The VLDB Journal* 30, 3 (May 2021), 1–22.

[9] Gabriel Haas, Michael Haubenschild, and Viktor Leis. 2020. Exploiting Directly-Attached NVMe Arrays in DBMS. In *10th Conference on Innovative Data Systems Research, CIDR 2020, Amsterdam, The Netherlands, January 12-15, 2020, Online Proceedings*.

[10] M. Hosomi, H. Yamagishi, T. Yamamoto, K. Bessho, Y. Higo, K. Yamane, H. Yamada, M. Shoji, H. Hachino, C. Fukumoto, H. Nagao, and H. Kano. 2005. A Novel Nonvolatile Memory with Spin Torque Transfer Magnetization Switching: Spin-RAM. In *IEEE International Electron Devices Meeting, 2005. IEDM Technical Digest. (IEEE IEDM Technical Digest)*.

[11] Intel Corporation. 2019. Optane DCPMM 100 Series Product Brief. Retrieved August 17, 2021 from https://www.intel.com/content/dam/www/public/us/en/documents/product-briefs/optane-dc-persistent-memory-brief.pdf.

[12] Intel Corporation. 2020. Intel Optane Persistent Memory Start Up Guide. Retrieved August 17, 2021 from https://www.intel.com/content/dam/support/us/en/documents/memory-and-storage/data-center-persistent-mem/Intel_Optane_Persistent_Memory_Start_Up_Guide.pdf.

[13] Intel Corporation. 2021. Intel Optane DC SSD Series. Retrieved August 17, 2021 from https://www.intel.com/content/www/us/en/products/details/memory-storage/data-center-ssds/optane-dc-ssd-series.html.

[14] Intel Corporation. 2021. Optane DCPMM 200 Series Product Brief. Retrieved August 17, 2021 from https://www.intel.com/content/dam/www/public/us/en/documents/product-briefs/optane-persistent-memory-200-series-brief.pdf.

[15] Intel Corporation. 2021. Optane SSD P5800X Series Product Brief. Retrieved August 17, 2021 from https://www.intel.com/content/dam/www/public/us/en/documents/product-briefs/a1169660-optane-ssd-p5800x-product-brief.pdf.

[16] Joseph Izraelevitz, Jian Yang, Lu Zhang, Juno Kim, Xiao Liu, Amir Saman Memaripour, Yun Joon Soh, Zixuan Wang, Yi Xu, Subramanya R. Dulloor, Jishen Zhao, and Steven Swanson. 2019. Basic Performance Measurements of the Intel Optane DC Persistent Memory Module. *CoRR* abs/1903.05714 (2019). arXiv:1903.05714

[17] Viktor Leis, Michael Haubenschild, Alfons Kemper, and Thomas Neumann. 2018. LeanStore: In-Memory Data Management beyond Main Memory. In *2018 IEEE 34th International Conference on Data Engineering (ICDE) (IEEE ICDE)*. 185–196.

[18] Alberto Lerner and Philippe Bonnet. 2021. Not Your Grandpa's SSD: The Era of Co-Designed Storage Devices. In *Proceedings of the 2021 International Conference on Management of Data* (Virtual Event, China) *(SIGMOD/PODS '21)*. 2852–2858.

[19] Lucas Lersch, Xiangpeng Hao, Ismail Oukid, Tianzheng Wang, and Thomas Willhalm. 2019. Evaluating Persistent Memory Range Indexes. *PVLDB* 13, 4 (Dec. 2019), 574–587.

[20] Jihang Liu, Shimin Chen, and Lujun Wang. 2020. LB+Trees: Optimizing Persistent Index Performance on 3DXPoint Memory. *PVLDB* 13, 7 (March 2020), 1078–1090.

[21] David Lomet. 2018. Cost/Performance in Modern Data Stores: How Data Caching Systems Succeed. In *Proceedings of the 14th International Workshop on Data Management on New Hardware (DaMoN '18)*. Article 9, 10 pages.

[22] Baotong Lu, Xiangpeng Hao, Tianzheng Wang, and Eric Lo. 2020. Dash: Scalable Hashing on Persistent Memory. *PVLDB* 13, 8 (April 2020), 1147–1161.

[23] Thomas Neumann and Michael J Freitag. 2020. Umbra: A Disk-Based System with In-Memory Performance. In *10th Conference on Innovative Data Systems Research, CIDR 2020, Amsterdam, The Netherlands, January 12-15, 2020, Online Proceedings*.

[24] Newegg. 2021. Intel Xeon Gold 6242R. Retrieved August 17, 2021 from https://www.newegg.com/p/0ZK-02J0-007D3.

[25] Newegg. 2021. Optane DC P4800X SSD. Retrieved August 17, 2021 from https://www.newegg.com/p/2U3-0001-00032.

[26] Newegg. 2021. Samsung 32GB Memory. Retrieved August 17, 2021 from https://www.newegg.com/p/9SIAFYREWP4976.

[27] Oracle. 2021. Using Asynchronous I/O on Linux. Retrieved November 21, 2021 from https://dev.mysql.com/doc/refman/5.7/en/innodb-linux-native-aio.html.

[28] 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. In *Proceedings of the 2016 International Conference on Management of Data* (San Francisco, California, USA) *(SIGMOD 16)*. 371–386.

[29] Ivan Luiz Picoli, Niclas Hedam, Philippe Bonnet, and Pinar Tözün. 2020. Open-Channel SSD (What is it Good For). In *10th Conference on Innovative Data Systems Research, CIDR 2020, Amsterdam, The Netherlands, January 12-15, 2020, Online Proceedings*.

[30] Samsung. 2021. Samsung 980 PRO SSD. Retrieved August 17, 2021 from https://www.samsung.com/semiconductor/minisite/ssd/product/consumer/980pro.

[31] Dmitri B Strukov, Gregory S Snider, Duncan R Stewart, and R Stanley Williams. 2008. The Missing Memristor Found. *Nature* 453, 7191 (2008), 80–83.

[32] Ben Walker and Jim Harris. 2019. 10.39M Storage I/O Per Second From One Thread. Retrieved August 17, 2021 from https://spdk.io/news/2019/05/06/nvme.

[33] Li Wang, Zining Zhang, Bingsheng He, and Zhenjie Zhang. 2020. PA-Tree: Polled-Mode Asynchronous B+ Tree for NVMe. In *2020 IEEE 36th International Conference on Data Engineering (ICDE)*. IEEE, 553–564.

[34] H. S P Wong, S. Raoux, SangBum Kim, Jiale Liang, John P. Reifenberg, B. Rajendran, Mehdi Asheghi, and Kenneth E. Goodson. 2010. Phase Change Memory. *Proc. IEEE* 98, 12 (2010), 2201–2227.

[35] Kan Wu, Andrea Arpaci-Dusseau, and Remzi Arpaci-Dusseau. 2019. Towards an Unwritten Contract of Intel Optane SSD. *HotStorage* (2019).

[36] Kan Wu, Zhihan Guo, Guanzhou Hu, Kaiwei Tu, Ramnatthan Alagappan, Rathijit Sen, Kwanghyun Park, Andrea C. Arpaci-Dusseau, and Remzi H. Arpaci-Dusseau. 2021. The Storage Hierarchy is Not a Hierarchy: Optimizing Caching on Modern Storage Devices with Orthus. In *19th USENIX Conference on File and Storage Technologies (FAST 21) (USENIX FAST)*. 307–323.

[37] Qiumin Xu, Huzefa Siyamwala, Mrinmoy Ghosh, Tameesh Suri, Manu Awasthi, Zvika Guz, Anahita Shayesteh, and Vijay Balakrishnan. 2015. Performance Analysis of NVMe SSDs and Their Implication on Real World Databases. In *Proceedings of the 8th ACM International Systems and Storage Conference (SYSTOR)*. Article 6, 11 pages.

[38] Jian Yang, Juno Kim, Morteza Hoseinzadeh, Joseph Izraelevitz, and Steven Swanson. 2020. An Empirical Guide to the Behavior and Use of Scalable Persistent Memory. In *18th USENIX Conference on File and Storage Technologies (FAST 20) (USENIX FAST)*. 169–182.