INVITED PAPER

Platform Storage Performance With 3D XPoint Technology

This paper reviews the potentialities on computing introduced by the 3-D XPoint technology in changing the memory-storage hierarchy.

By Frank T. Hady, Annie Foong, Bryan Veal, and Dan Williams

ABSTRACT | With a combination of high performance and nonvolatility, the arrival of 3D XPoint memory promises to fundamentally change the memory-storage hierarchy at the hardware, system software, and application levels. This memory will be deployed first as a block addressable storage device, known as the Intel Optane SSD, and even in this familiar form it will drive basic system change. Access times consistently as fast, or faster, than the rest of the system will blur the line between storage and memory. The low latencies from these solid-state drives (SSDs) allow rethinking even basic storage methodologies to be more memory-like. For example, the manner in which storage performance is measured shifts from input–output operations (IOs) at a given queue depth to response time for a given load, like memory is typically measured. System changes to match the low latency of these SSDs are already advanced, and in many cases they enable the application to utilize the SSD's performance. In other cases, additional work is required, particularly on policies set originally with slow storage in mind. On top of these already-capable systems are real applications. System-level tests show that applications such as key–value stores and real-time analytics can benefit immediately. These application benefits include significantly faster runtime (up to 3×) and access to larger data sets than supported in DRAM. Newly viable mechanisms for expanding application memory footprint include native application support or native operating system paging, a significant change in the use of SSDs. The next step in this convergence is 3D XPoint memory accessed through processor load/store operations. Significant operating system support is already in place. The implications of consistently low latency storage and fast persistent memory on computing are great, with applications and systems taking advantage of this new technology as storage as the first to benefit.

Manuscript received January 16, 2017; revised June 16, 2017; accepted July 17, 2017. Date of publication August 7, 2017; date of current version August 18, 2017.
(Corresponding author: Frank T. Hady.)
The authors are with Intel Corporation, Hillsboro, OR 97124 USA (e-mail: frank.hady@intel.com; annie.foong@intel.com; bryan.e.veal@intel.com; dan.j.williams@intel.com).

Digital Object Identifier: 10.1109/JPROC.2017.2731776

KEYWORDS | Big data applications; computer architecture; computer performance; database systems; distributed databases; flash memories; Intel Optane SSD; memory architecture; nonvolatile memory; persistent memory; solid-state drives; system software; 3D XPoint memory

I. INTRODUCTION

For decades, storage has been the performance laggard in the computer system. Processors have consistently increased in performance year after year, while data on the hard drive remained milliseconds away. Much system research and engineering innovation has been devoted to working around this latency divide. Recently, NAND-based solid-state drives (SSDs) have improved over hard drive latencies by more than 10×. Even with NAND SSDs, the fundamental system balance is unchanged—storage has remained the latency laggard. New memory technology, often called storage-class memory, promises to upend this decades-old disparity. Deployed as SSDs or as persistent memory, this new technology makes persistent data storage as fast as the rest of the system, providing new opportunities for fundamental system changes. Included in this class of technology, at various stages of maturity, are PCM [1], MRAM [2], ReRAM [3], NVDIMM [4], and 3D XPoint memory. In this paper, we focus on the system impact of Intel Optane SSDs, featuring 3D XPoint memory, which recently began shipping.

The authors are part of the team responsible for 3D XPoint technology systems architecture. When we began our work in 2008, with systems still in transition to NAND SSDs, the prevailing belief was that even if new lower latency SSDs were realized, ingrained system software and hardware practices would mask application benefit. Many researchers have focused on pushing revolutionary new system and device designs [5]–[7]. However, armed with the knowledge of much lower latency SSDs in development, our focus has been evolutionary system changes that modify systems only where needed while still

making the lower latency storage performance advantage available to system applications. Through a series of system-level analyses that have pinpointed bottlenecks in the storage path, we have provided empirical evidence to guide optimization to ensure that systems hardware and software will not bottleneck low-latency SSDs. Here we will show that the system is ready for applications to effectively access the benefits of low latency storage, both as storage and as extended memory.

In Section II, we explore the system configurations enabling the three major usage models for storage-class memory: persistent storage, nonpersistent DRAM extension, and persistent memory. We show how both low-latency SSDs and persistent memory devices can enable all three models.

In Section III, we summarize historical system hardware and software analyses and optimizations which allow today's systems to take advantage of high-performance NAND-based PCI Express SSDs. This provides the baseline for performance evaluation and further systems enabling for Intel Optane SSDs.

So that we can adequately evaluate low-latency SSD performance, in Section IV, we explain why we must move away from the common input–output operations per second (IOPS)-at-queue-depth storage performance measurement methodology to a response-time-based memory performance measurement methodology.

In Section V, through benchmark and workload characterizations, we provide examples of workload performance and data-set-size advantages using Intel Optane SSDs.

In Section VI, we introduce systems enabling already underway to enable 3D XPoint memory deployed as a memory module.

II. 3D XPOINT MEMORY AS STORAGE AND MEMORY

The primary focus of this paper is 3D XPoint memory accessible as an Intel Optane SSD. We discuss such an SSD in the context of the system, the use model, and in the context of persistent memory. Three predominant use models exist within the system for this new memory.

1) *Storage:* Persistence is required. Applications use the services of an operating system and file system, or use self-implemented code to manage policies and ensure transactional correctness. It is generally delivered as an SSD, made usable via a software-managed transaction, using the block abstraction that systems have come to expect from disk-based storage for decades. Applications access storage with system calls such as `read()` and `write()`, which perform asynchronous input–output operations (IOs) to the device.

2) *DRAM extension:* Persistence is not expected. This use case is typically motivated by engineering limits or cost limits to the total capacity of DRAM. Applications address objects as memory with CPU load and store instructions, while the operating system moves data between memory and storage via paging.

3) *Persistent memory:* Persistence is required. Applications address objects as memory which is capable of maintaining persistence requirements. Persistent memory is usable directly via processor load and store instructions. Although not explicitly stated, to be broadly useful, a persistent memory device must necessarily have latencies commensurate with processor speeds.

A. Storage

Experience tells us that with any new technology, there will be customers who seek to use new technologies quickly by maximizing backward compatibility (e.g., by continuing to use `read()` and `write()` system calls) at the cost of some performance. To this end, 3D XPoint memory will be used first as faster storage in an Optane SSD [Fig. 1(a)].

The Optane SSD has been optimized throughput to deliver low IO latency. We will show the impact of the SSD's NVMe interface later in the paper. Within the SSD itself, the controller spreads single 4-kB IOs across multiple 3D XPoint memory channels to harness the throughput of multiple memory dies to deliver low latency for a single IO. Contrast this with NAND-based SSDs which generally access a single die to satisfy a 4-kB (or larger) IO. The Optane SSD includes a hardware-only path for normal reads and writes from host to media, avoiding the added latency from firmware often seen in NAND SSDs. Using 3D XPoint memory, the Optane SSD is also able to write data in place, avoiding the erase–write-driven garbage collecting required for NAND SSDs. As we will show, this combination of features enables the Optane SSD to deliver very low average latency with much smaller outlier latencies than NAND SSDs.

Like an SSD, it is also possible to use persistent memory as storage, enabling existing applications to execute unmodified, by using an operating system that supports persistent memory block device emulation. An example of block emulation is the persistent memory block device support up-streamed in Linux kernel 4.1 [8]. New file systems [6], [9], [10] have emerged to make full use of the byte-addressable capability of persistent memory, allowing applications to reuse familiar file semantics. Dulloor [11] has created a file system to enable standard file operations to persistent memory, enabling significant performance advantages for the persistent memory file system.

B. DRAM Extension

Additionally, even in the form of an SSD, 3D XPoint memory can be used effectively to extend DRAM via paging (also called swapping) [Fig. 1(b)] without requiring applications to change. Paging to extend DRAM had been a mainstay of operating systems' virtual memory management. We have found that paging has worked surprisingly

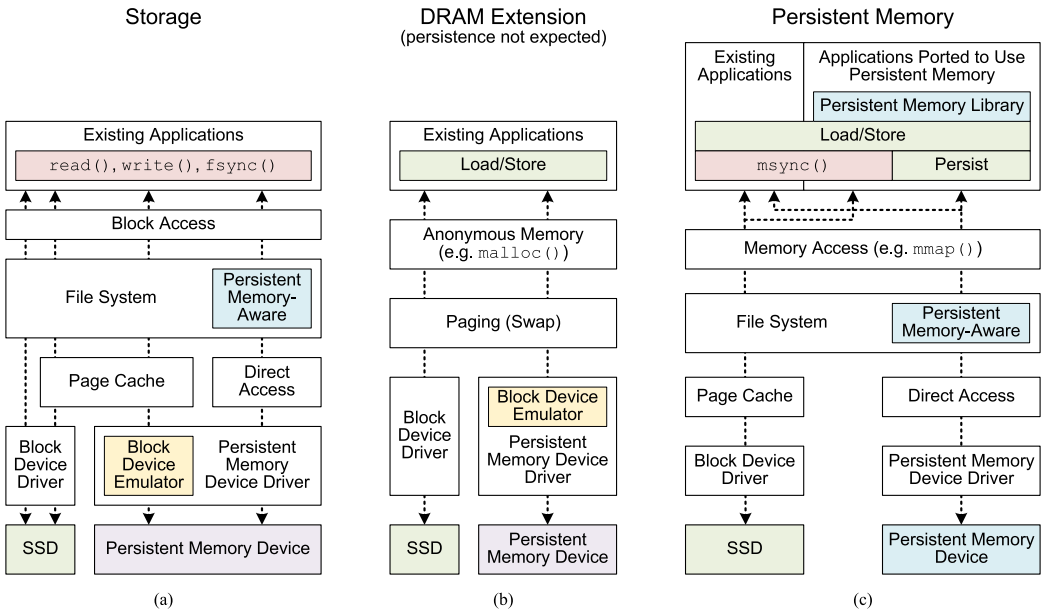

Fig. 1. ***Possible use models for SSDs and persistent memory devices: (a) as storage; (b) as DRAM extension; or (c) as persistent memory.***

well with Optane SSD low latency since Linux* kernel 3.18 (with some policy parameter tuning). A hypervisor may also provide paging, providing an avenue for optimization without tuning or modifying the operating system. [12].

C. Persistent Memory

Applications already using memory-mapped storage (e.g., via `mmap()`) that commit writes when needed (e.g., via `msync()`) are already effectively persistent memory-aware [Fig. 1(c) (left)]. No code changes will be needed of the applications or operating system for correctness in a persistent memory. Performance gains will be workload dependent, but applications will work correctly. We will return to this topic, with quantification, in Section III-B.

As a step toward native persistent memory support, operating systems have added drivers for persistent memory devices [13], allowing higher layers to build upon them [Fig. 1(c) (right)]. The Linux kernel has integrated the direct access block layer (DAX) as an optimization for persistent memory. In turn, DAX support has been added to the Ext4 and XFS file systems [Fig. 1(a) (right)]. Using DAX, an unmodified application can reap immediate benefits of persistent memory versus full block device emulation [14]. Traditional `read()` and `write()` system calls on DAX-enabled files bypass the operating system's superfluous page cache and are instead implemented as memory copies to persistent memory.

Similarly, for applications that utilize memory-mapped files or devices, DAX again removes the page cache by mapping persistent memory directly into the application's virtual address space, where it can be accessed directly with load and store CPU instructions. However, persistence requires using additional instructions to flush CPU caches to commit data to persistent memory.

To accommodate existing applications, the requirement of persistence drives persistent memory to be managed similarly to file systems [13]. However, persistent memory can also be enabled via helper libraries that provide higher level functionality to ease application porting. NVM library [15], for example, is a collection of seven libraries, aimed at enabling use cases including transactional object stores, memory pool management, and accessing remote persistent memory via RDMA protocols.

III. A DECADE OF OPTIMIZATIONS AND RELATED WORK

In 2008, we were aware of the possibility of SSDs with an order of magnitude faster response time than the NAND SSDs, and so began to investigate the optimizations required in the rest of the system to effectively access such SSDs. We began by measuring system level latencies and overheads. Throughout we have used Linux on an Intel Xeon processor-based server platform as the reference system to maintain a common baseline with the flexibility to implement changes as needed. We believe (and have empirically validated) that the architectural insights gained apply generally to other system software. A minimal storage path consists of storage software, drivers, storage protocols, and controllers (Fig. 2).

We instrumented the entire storage path (using code instrumentation for software, PCI Express* (PCIe*) and SATA analyzers for hardware). We examined the time profile of an IO from an application to a SATA SSD, on a platform and operating system available in 2008 [16]. At the time, latency was still dominated by the storage device. A NAND SSD took about 140 μs, while the rest of system hardware and software—everything from storage interconnect to the application—was responsible for 40 μs. As we show

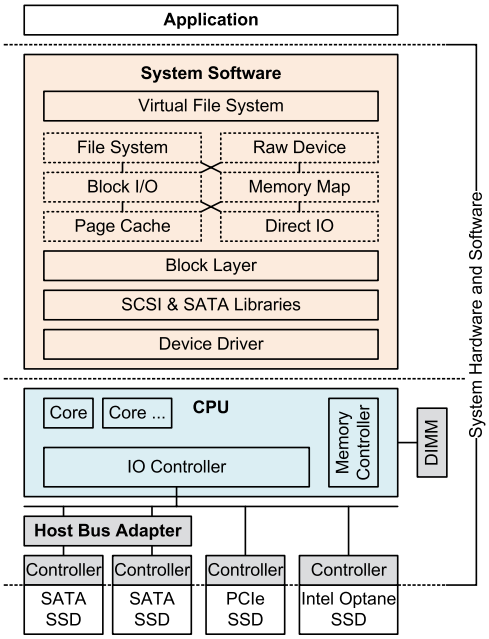

Fig. 2. ***An example of system hardware and system software contributing to IO delays.***

later in this section, while the system latency was not the bottleneck in 2008, it would have been for an Optane SSD. With this knowledge, we proceeded with three architectural goals: 1) reduce system latencies; 2) scale throughput; and 3) improve systems software compute efficiency (if needed).

A. Platform Hardware Optimization

We observed that SATA or SAS as a storage attach point added significant latency overheads to the storage path. A host bus adapter (HBA) is required to convert from PCIe (the ubiquitous CPU–IO system interconnect) to a SAS or SATA protocol. Such an HBA added about 25 μs of latency to the storage path. Additionally, a primary role of the HBA is to aggregate multiple low bandwidth storage devices (hundreds of megabytes per second) to match the upstream system interconnect bandwidth (gigabytes per second). This makes sense for slower storage subsystems, but not for low response time and/or high bandwidth storage. An optimal interface for a low-latency SSD must necessarily be one that removes the HBA and connects directly to the CPU.

Attaching SSDs directly to the PCIe system bus also offers bandwidth scalable with CPU supportable PCIe lanes. Amdahl's Balanced System Law [17] posits that a system needs one bit of IO per CPU instruction. A modern server system is capable of more than 100 GHz of compute. Assuming a CPI (clocks per instruction) of 1, this system would therefore require 12.5 GB/s of sequential IO bandwidth, or three million IOPS of 4-kB random-access operations. This translates to 20–40 SATA or SAS SSDs, or 20 000–30 000 hard drives (assuming random access). A modern server processor is capable of supporting this IO demand by directly exposing more than 40 GB/s of PCIe lanes. The better choice to achieve Amdahl's balance is just four 4-GB/s PCIe SSDs—enabling deployment ease, lower power, and lower cost.

Early versions of PCIe SSDs appeared in the market in 2007 [18]. Recognizing that an efficient, standard interface was needed, Intel worked with the industry to create the NVM Express* (NVMe*) specification for PCIe SSDs [19].

B. System Software Optimizations

In 2008, a typical storage software stack (e.g., Microsoft* Windows* operating system or Linux operating system) incurred about 12 μs of processing [16]. With hard disk drives, compute requirements of the software stack were never an issue. With a million-IOPS SSD, the compute requirements to support one SSD would have saturated ten processor cores. A commodity server hosts about 20–40 cores, and a commercial 2U server chassis is capable of housing up to 24 2.5-in SSDs. Fully using this many required improving the processing efficiency of the storage stack to ensure that the compute requirements of storage do not bottleneck the system. Our analysis motivated reducing this compute requirement in two ways: 1) to design a controller interface that pushes IO completions instead of relying on host-initiated status register reads such as those used by the common Application Host Controller Interface (AHCI); and 2) to enable performance to scale with cores by directing IO completion processing to the requesting core (via multiple queues and vectored interrupts). Both proposals were adopted by NVMe.

With these optimizations, our analysis showed that processing would be equally divided between the file system, IO block and protocol processing, context switching, and drivers. Linux NVMe driver developers avoided the usual SCSI/ATA protocol layer and instead directly joined the block and driver layers into a highly optimized storage stack for NVMe SSDs. Bjørling's blk-mq (multiqueue block layer) patch [20] reduced the number of instructions and contention in the IO stack for block storage, which reduced the processing requirement of Linux storage stack from 9 to 4 μs. Yang *et al.* [21] further pushed the envelope by removing context switches from the path through polling, showing the empirical minimum possibility of 1.5 μs for a storage stack.

Given that many applications access storage via a file system, we also examined the overheads of the Ext3 file system and found that the code path had a relatively small overhead of 1.5 μs (not shown). However, we saw that policies and metadata layouts were based on hard disk assumptions (i.e., optimized for sequential access) and had a greater impact on performance. At the time, there were already multiple efforts underway to design NVM-aware file systems [6], [22], [23].

Recently, Sehgal *et al.* [24] validated our earlier qualitative insights with a comprehensive empirical comparison of NVM-aware file systems (PMFS and F2FS) against five traditional file systems (Ext2, Ext3, Ext4, XFS, and NILFS2). They showed that by selecting the right policies—journaling modes, allocation policies, and avoiding the buffer cache with execute-in-place—existing file systems approach within 5% of PMFS for real-world workload benchmarks. Countering this, more recent file systems such as NOVA [25] exceed the performance of PMFS.

Finally, we looked to validate the viability of supporting a DRAM-extension use model through paging [Fig. 1(b)]. Historically, paging was not a viable option because of high storage latency, even with NAND SSDs. Yang [26] showed that with Linux kernel 2.x, hard-drive centric paging policies resulted in high outliers (>60 ms). Our analysis (since Linux kernel 3.18) showed that with policy tuning, the code to implement paging incurred overheads of only 5 μs. The default policy to always prefetch and swap out multiple blocks may be optimal for hard drives, but not so for Optane SSDs.

Measured on Linux kernel 4.6.7 using a single core, a page miss requires on average 92 μs for a NAND SSD, but only 16 μs for an Intel Optane SSD.[1] To improve multicore performance scaling of paging, Linux developers are working to remove sources of contention [27]. We will show the viability of using Intel Optane SSD to extend DRAM for a real-world workload in the next sections.

Beyond the code inefficiencies noted, we have found that system software and hardware mechanisms are nearly optimal, including file systems and paging implementations. We have exposed latency outliers due to policies that optimize for slow storage, e.g., interrupt coalescing in drivers, IO merging in IO schedulers, prefetching, using log structures instead of in-place updates, and using an intermediate buffer cache. Such policies make sense for hard drives and even NAND SSDs. However, for Intel Optane SSDs, the system balance point has changed; sophisticated IO scheduling and buffering are not only unnecessary but get in the way. Often, these policy changes are simple to implement, offering the best return in optimization investment.

Fig. 3 shows the impact of much of the cross-industry work described in this section by plotting IOPS versus system latency for three different platform interfaces—SATA HBA, SAS HBA, and PCIe SSD—all with 4-GB/s PCIe links to the host.[2] Note that only system software and hardware latency and throughput are shown, excluding media latency, to focus on system impacts. For the SATA and SAS data points, we attached as many SATA NAND SSDs as needed to saturate one SATA or SAS HBA to provide a fair comparison. The combination of systems software and hardware optimizations enabled an NVMe interface to reach near PCIe theoretical maximum (about 800 000 4-kB IOPS) while maintaining significantly lower latency until maximum IOPS.

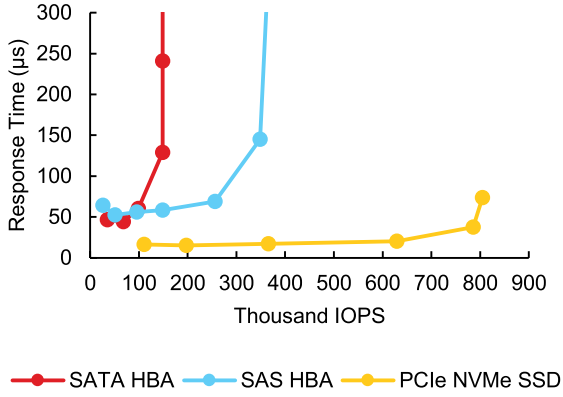

Fig. 3. ***99.99th percentile response times (excluding media) plotted against IOPS for 4-kB reads for three SSD interfaces.***

Finally, we combine a well-optimized system with 3D XPoint memory resulting in the Intel Optane SSD latencies shown in Fig. 4. For comparison, we included data for the SATA SSD system from 2008, and a NAND NVMe SSD. The time to access a 4-kB block from an Intel Optane SSD, measured at the PCIe bus from NVMe doorbell write to completing interrupt, is less than 8 μs, showing the result of the earlier discussed latency optimizations built into the Optane SSD. The time from application to the SSD in total about 11 μs, revealing that the system added less than 4 μs. Systems (when correctly configured or optimized) have exposed ultralow latency storage benefits to applications.

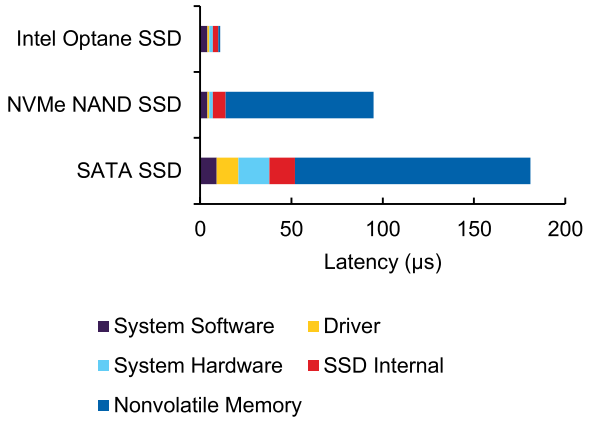

Fig. 4. ***Latency from application to device for an Intel Optane SSD as compared to NAND SSDs.***

[1]Measured on a single-processor 3.4-GHz Intel Core i7-4770 platform.

[2]Measured on a 2.3-GHz two-processor Intel Xeon E5-2695 platform, with the integrated AHCI SATA HBA, Broadcom* SAS 9207-8I HBA, and Intel SSD DC P3700 Series.

IV. EVALUATING PERFORMANCE

As we worked to tune the system, experimenting with low latency storage, it became apparent to us that the way we measured storage needed to change. Measurement of storage device performance has a long history with an established language and set of measurement methodologies. While traditional methodologies have served for measuring hard disk drives and NAND SSDs, in this section, we show how these queue-depth-based methodologies fail to accurately compare traditional SSDs with those whose latencies are similar to the software stack used to access them. We introduce an improved methodology based on IO response times, mirroring platform memory subsystem measurement.

Today's common storage performance metric is IOPS. Storage benchmark tools like fio [28] and Iometer [29] are typically used to measure IOPS. These tools run a specified number of threads, with each thread issuing IOs to the device. The total number of concurrently outstanding IOs the tool seeks to generate is called the queue depth of the test. In common performance benchmarking methodology, such as "four-corner testing" used by Tom's Hardware [30], queue depth is configured to manipulate the load on the SSD, while IOPS are measured. As such, queue depth is reported as the independent variable on the x-axis, while IOPS becomes the dependent variable on the y-axis (Fig. 5).

High-capacity NAND SSDs have many NAND dies, and rely on the parallel distribution of IOs across the dies to overcome the per-die latency and deliver high IOPS [31]. In performance testing described previously, high queue depths are used to provide IOs at a rate high enough to utilize the parallelism benefits. As shown for the NAND-based Intel SSD DC P3700 Series in Fig. 5, IOPS increase as queue depth is increased.[3]

For the Intel Optane SSD DC P4800X Series, Fig. 5 shows a rapid increase in IOPS resulting from the Optane SSD's low latency media and earlier described design, culmination in maximum IOPS at queue depth of only 8. This is a clear improvement; for example, the NAND SSD reaches about 300 000 IOPS at a queue depth of 128, while the Intel Optane SSD reaches a similar IOPS at a queue depth of 4. However, using queue depth to represent load inadequately exposes the performance improvement of low latency storage like the Intel Optane SSD.

Fig. 2 showed the three contributors to the processing time per IO when measured from a user-level application: system software, system hardware, and the SSD itself. The average number of outstanding IOs at each of these layers is proportional to the layer's delay. Historically, almost all of this time was taken at the storage device itself, so queue depth was a good proxy for load on the device. For SSDs with storage-class memory, delays in the SSD are comparable to system delays. This makes queue depth a poor representation of SSD load, and therefore a poor basis for comparison.

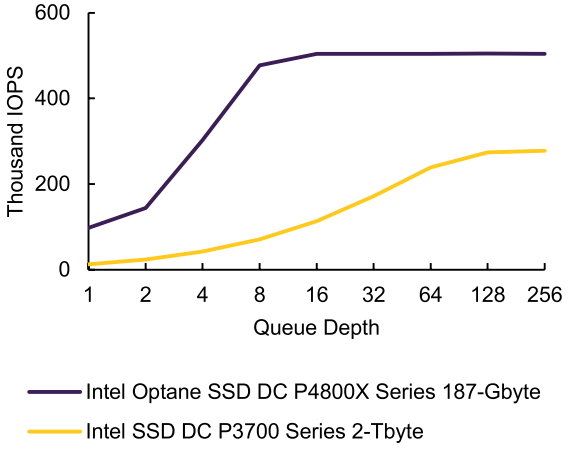

Fig. 5. ***A comparison of IOPS at given queue depths between a NAND SSD and an Intel Optane SSD, given a randomly distributed workload of 4-kB IOs with 70% reads and 30% writes.***

An example is provided in Fig. 6. Consider a workload that requests an IO every 20 μs issued on a system that adds a 10 μs of delay to every IO. For an SSD with a 100-μs delay, this workload results in an average queue depth of 5 at the SSD [Fig. 6(a)]. For a second SSD with a lower delay of 10 μs, the average queue depth at the SSD is 0.5 [Fig. 6(b)]. The same system load results in a much lower queue depth at the SSD. Half the time, the second SSD is left with no work.

To attempt a fairer comparison between the two SSDs, the workload could be increased to an IO every 4 μs to achieve an overall average queue depth of 5 for the second SSD. However, now the SSD itself sees a queue depth of only 2.5 since the system load also increases [Fig. 6(c)]. Such an attempt to patch the methodology and produce

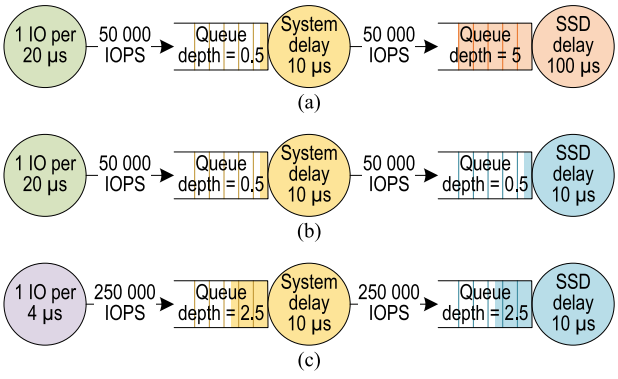

Fig. 6. ***A comparison of average queue depths for systems with (a) a higher delay SSD; (b) a lower delay SSD; and (c) a lower delay SSD under a higher request rate.***

[3]Performance was measured on system with two Intel Xeon E5-2699 v4 processors 2.2 GHz with Intel Turbo Boost Technology, up to 3.6 GHz; an Intel Server Board S2600WT Family; 256 Gbytes DRAM; Ubuntu* Server operating system 16.04.2 LTS x86_64; Linux kernel 4.4.0–21–generic (Ubuntu build); and fio 2.2.10. SMP interrupt affinity was changed, with irqbalance disabled, for the SSDs. The CPU governor was set to performance, and udevd was disabled.

equal queue depths does not result in a fair performance comparison.

Perhaps more importantly, IOPS at queue depth ignores wait time for the data to be retrieved. The delay itself is the performance metric that provides the most relevant comparison. Fig. 7 shows a comparison of latency, or delay given a minimal load, between the NAND-based SSD and the Intel Optane SSD. On average, we see that the NAND SSD incurs about 10× the latency of the Intel Optane SSD. Moreover, what is shown in Fig. 7, which cannot be shown in a queue-depth-versus-IOPS plot, is the large difference in latency outliers between the two SSDs, up to the 99.999th percentile. We consider lower delay outliers to be a higher quality of service (QoS). In Section IV, we will show workloads that benefit from higher QoS.

Expanding from latency, Fig. 8 shows the delay under load, or response time, and the QoS of response times, for the same tests used to produce Fig. 5. Here, load is directly represented by IOPS, regardless of the queue depth used to produce the load, and the response time is presented as the performance metric. On the log-scale *y*-axis, response time for the NAND SSD increases as the load increases. For the Intel Optane SSD, the curve is reminiscent of system memory measurements, where response time remains low, and only rises significantly when the load saturates the SSD's bandwidth.

Moreover, Fig. 8 shows a higher QoS for response times for the Intel Optane SSD. Even under heavy load, the 99.999th percentile response time remains under 100 μs. On the other hand, for the NAND SSD, the 99th percentile response time exceeds 1 ms, while the 99.999th percentile response time approaches 10 ms.

We contend that our methodology of plotting response time versus throughput, regardless of queue depth, including QoS for high-percentile outliers, is the right way to measure and compare SSD performance. In fact, systems memory performance is already historically reported as

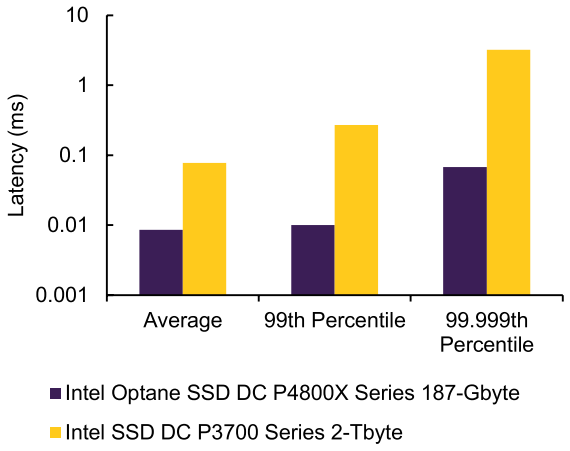

Fig. 7. ***A comparison of average latency and latency QoS between a NAND SSD and an Intel Optane SSD given a randomly distributed workload of 4-kB IOs with 70% reads and 30% writes.***

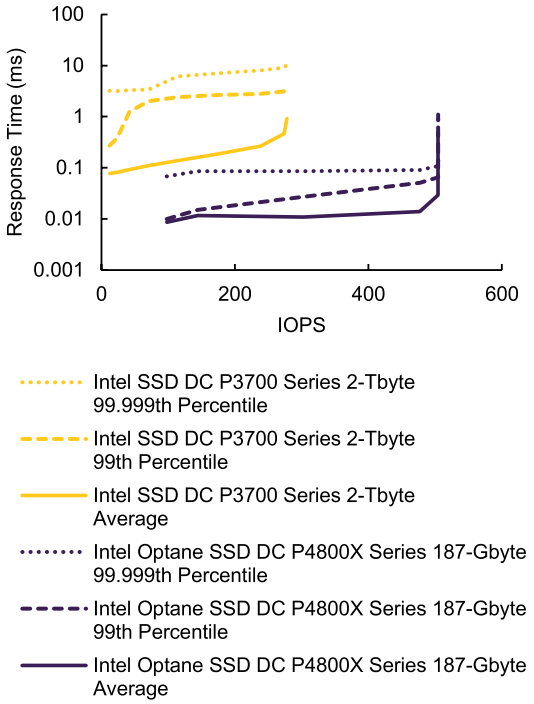

Fig. 8. ***A comparison of average, 99th percentile, and 99.999th percentile response times at given IOPS, between a NAND SSD and an Intel Optane SSD given a randomly distributed workload of 4-kB IOs with 70% reads and 30% writes.***

response time under a given load [32]. We urge others to adopt this methodology for storage. Measuring response time versus IOPS will enable systems experts and application designers to directly gauge how a given SSD's average and outlier response times will affect performance of the workload they intend to place on the system.

V. USE CASES AND REFERENCE APPLICATIONS

With the system components understood, we turn to characterizing the impact on real use. The two most practical use models for initial Intel Optane SSD deployments are: Faster storage and DRAM extension.

A. Faster Storage for Databases: RocksDB

Over the past decade databases optimized for high-performance multicore systems featuring NAND-based storage have emerged. RocksDB [33] is an open source example of such a database. This embeddable database uses log structured merge (LSM) trees to provide a high-performance embeddable key–value store. The embedded LSM tree drives much of the storage level behavior of the application. At the SSD, reads are often randomly addressed while writes are generally sequentially addressed and there are more reads than writes. Satisfying a single user request for a web page requires many key–value retrievals. Storage in such a

system should return many values quickly, meaning it must deliver high IOPS for mixed read/write workload described, so many user requests can be satisfied quickly with a single SSD. Storage must also deliver these IOs quickly with great consistency of delay so individual users are not required to wait long for a response. In other words, the SSD should deliver high IOPS and excellent QoS for mixed random reads and sequential write workload.

An Intel Optane SSD prototype (a precursor to the Intel Optane SSD DC P4800X with very similar performance) was tested versus a NAND SSD (Intel SSD DC P3600 Series) in a system with two Intel Xeon processors (2.5 GHz, 12 cores each, with Intel hyper-threading technology enabled), 256 GB of DDR4, CentOS* Linux distribution 7.2, using XFS with no operating system changes. TRIM was enabled. The Intel SSD DC P3600 Series was filled to 50% capacity, while the Intel Optane SSD prototype was 75% filled. RocksDB was setup based on published tests at rocksdb.org: a 1 billion-key database, eight "shards" with 25 million key–value pairs each, 20-B keys, 800-B values, 50% compression, about 100 GB on-disk. Read: All threads randomly read all keys. Read/Write: Threads randomly reads keys while one writer thread updates up to about 80 000 keys/s. The results of the experimental runs are included in Figs. 9 and 10, both displaying performance versus number of RocksDB threads.

User throughput (Fig. 9) shows a substantial advantage for the Intel Optane SSD prototype. For this experiment, number of threads roughly corresponds to number of cores. The much lower latency of the Intel Optane SSD prototype results in significantly higher IOPS for read-only workloads since there are not enough outstanding accesses to fully use the die-level parallelism of the NAND SSD. The coupling of compute time per access, and the latency of the SSDs results in a 3× advantage in throughput for the Intel Optane SSD prototype.

As important as throughput is user response time. For this metric, the tail of the response time distribution of the SSD must be considered, so Fig. 10 shows the 99th percentile response time. Latency in this chart represents a component of response time to the end user. The workload is a particularly difficult one for NAND-based SSDs since writes take a long time to complete on NAND dies and some random reads may find themselves waiting for a write to complete. 3D XPoint memory completes writes much more quickly so the Intel Optane SSD prototype does not suffer as much collision penalty. Note that for all thread counts the Intel Optane SSD prototype delivers requests in 1/10th the time required by the NAND-based SSD, resulting in a 10× better user responsiveness.

RocksDB is an important application in its own right, but it is also representative of a very important class of applications, key–value stores, which are sensitive to random read latency in the presence of background sequential writes. As shown, the Intel Optane SSD prototype provides a significant throughput and responsiveness advantage versus NAND SSDs for this very important workload.

B. Faster Storage for Real Time Analytics: Aerospike ACT

Aerospike* is an enterprise-class NoSQL database coded with NVM and SSDs in mind [34], [35]. Aerospike targets mission-critical, real-time analytics workloads, such as ad-bidding and fraud prevention. Such use cases have stringent response time requirements while concurrently reading and writing. Aerospike provided the Aerospike Certification Tool (ACT) [36] to enable users a better method of rating SSD performance. ACT is specifically designed to ensure that an SSD device can sustain the high volume of transactions required by high-performance real-time databases, yet passing high QoS expectations of real time applications. While ACT is a microbenchmark and not the actual Aerospike database, it represents the specific workload characteristics that Aerospike (the business) sees as representative of their database's performance.

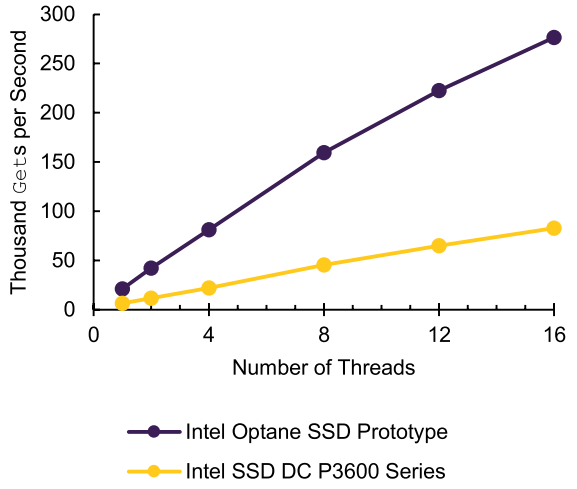

Fig. 9. ***Throughput for RocksDB Get operations.***

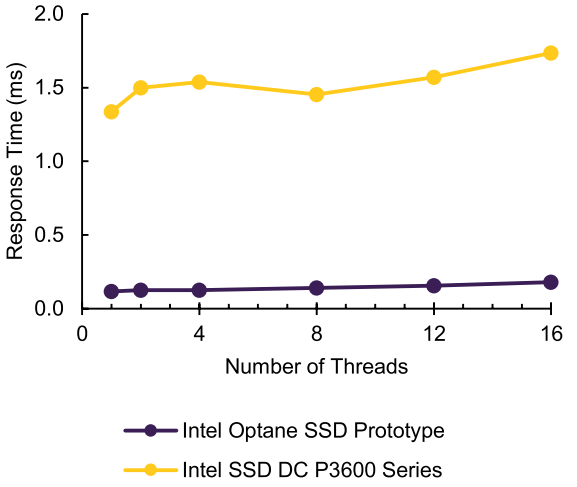

Fig. 10. ***99th percentile response times for RocksDB Get operations.***

ACT performs a combination of large (128-kB) block reads and writes and small (1.5-kB) block reads, simulating standard real-time database read/write loads. ACT measures the response time of small read requests (which emulate database lookups), which occur in the presence of large read and write requests (which emulates the background garbage collection process of Aerospike's log-structured file system). The baseline ACT workload consists of the following:

1) 1000 IOPS (1.536 MB/s) of randomly distributed 1.5-kB reads (measured);
2) 1.536 MB/s of 128-kB reads;
3) 1.536 MB/s of 128-kB writes.

ACT reports the QoS of response times for the randomly distributed reads once per hour for 48 h. For a passing score, the following must hold for each 1-h interval:

1) 95th percentile response time ≤1 ms;
2) 99th percentile response time ≤8 ms;
3) 99.9th percentile response time ≤64 ms.

ACT's workload increases the throughput of both measured and background IO requests until the point of failure by incrementally applying a workload multiplier to the baseline workload. The largest multiple that produces a passing grade for a given SSD is the metric of goodness in comparing the performance of SSDs—the bigger the better.

Our tests compared the performance of an Intel Optane SSD prototype to a NAND-based PCIe 1.6-TB Intel SSD DC P3700 Series and a NAND-based SATA 400-GB Intel SSD DC S3700 Series.[4] ACT was configured to use eight queues with eight threads per queue and the tests ran for 2 h. (Official tests require 48 h; we used a shorter run for expediency as we have validated previously that the length of the run did not affect results for the SSDs under test.) With the standard ACT test, the Intel Optane SSD achieved a 150× workload multiplier, which is a 3.75× improvement over the NAND-based SSD (Fig. 11). At this multiplier, ACT has effectively saturated device throughput while maintaining high QoS.

We found that even with ACT uncompromising requirements, it had specified the highest level of QoS expectation as 95th percentile response time ≤1 ms. We knew from our microbenchmarks that we could do better. We tightened the QoS criteria further, beyond what is required in ACT, to 95th percentile response time ≤ 128 μs. With this new criterion, the Intel Optane SSD achieved a 125× workload multiplier over the NAND-based SSD 8× workload multiplier, yielding an improvement of 15× (Fig. 11).

[4]Measured on CentOS Linux release 7.1.1503, Intel Core i7 processor 4770, ASUS* H87I-PLUS motherboard, and 4-GB DDR3 memory.

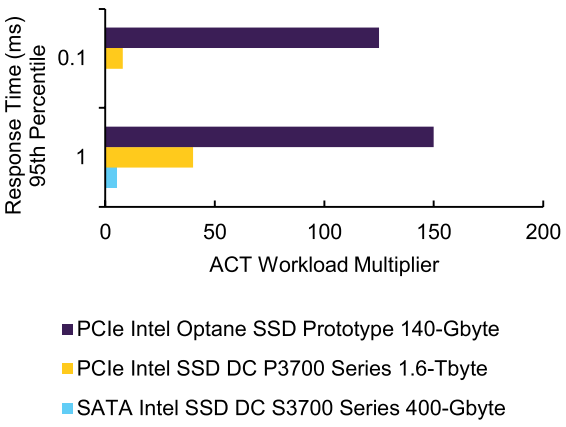

Fig. 11. ***Highest achieved ACT workload multipliers for three SSDs given 95th percentile response times of 1 and 0.1 ms.***

C. Extending DRAM for Memcached

Many modern applications are written with the expectation that the entire data set fits in memory. Amdahl's Memory Law [17] posits that the number of megabytes of DRAM needed to support 1 MIPS, or α, is one. Gray and Shenoy [37] observed the need to revise α to 4–10 for new data-centric workloads. A modern system would require upwards of 1 TB of DRAM. We deduced from our earlier paging microbenchmarks that paging is a compelling option with low latency SSDs.

To determine if selected reference applications can reap such benefits we chose memcached, an open-source in-memory key–value distributed cache that is widely deployed [38]. Memcached enables access to a caching server across the network. We used what we believed to be a best-case workload—random 4-kB get commands (reads).

Empirical measurements are shown in Fig. 12. Because these measurements predated the availability of Intel Optane SSDs, we substituted a nonproduction DRAM-based

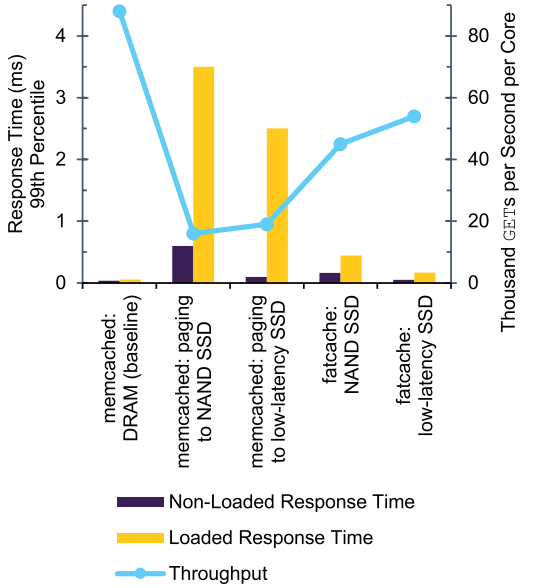

Fig. 12. ***Response time and throughput performance of memcached and fatcache with different devices.***

NVMe SSD with similar latency to the Intel Optane SSD while maintaining the same system software and hardware overheads. We have compared the performance of the two, and given the idyllic response curves of the Optane SSD, we are confident the results still apply.

The response time of paging to fast storage is 45× longer than memcached using DRAM, at less than 20% of throughput.[5] This suboptimal performance came as a surprise to us.

On root causing, we discovered that the issue is due to generic, context-unaware operating system paging. Since memcached is written with DRAM in mind, memcached updates caching statistics (temporal data structures, e.g., last modified time), on every access with small sized writes, for memcached get commands (reads) or set commands (writes). As such, pages are dirtied and must be written back to disks even on memcached reads.

Memcached represented a class of applications that require code modification to truly make use of paging to low latency storage. We needed an application that is written with temporal data structures separated from data. We found an open-source variant of memcached, called fatcache, written with SSDs in mind [39]. Fatcache organizes its metadata and data into separate buckets of storage block-sized units, and it manages its own disk IO rather than rely on paging provided by the operating system. With a low-latency SSD, fatcache achieves at least 60% of the throughput of memcached with DRAM alone.

Fatcache still incurs storage software processing overheads. We used a single core to accurately measure this impact while expecting that use of multiple instances of fatcache, running in parallel across multiple cores, will scale throughput further. Fatcache incurs an additional 10 μs of latency on an unloaded system, and an extra 110 μs at 99th percentile. Given that the network round trips in a data center range from 300 μs to 1.2 ms, we believe that distributed cache workloads can tolerate the extra latency. For this use case, a low latency SSD is a viable and less expensive option than additional DRAM to enable a larger data set size for the application.

VI. OPPORTUNITIES AHEAD

Even with the aggressive system optimizations done to date by many, the platform storage stack hardware/software still introduce microseconds of latency to the time to access 3D XPoint memory. Additionally, storage accesses have a minimum granularity of 512 B (often 4 kB in practice) further increasing the latency when accesses required only small data sizes. There is more performance advantage to be had with 3D XPoint memory by enabling a persistent memory usage model with additional systems hardware and software system changes.

[5]Measured on a two-socket 2.9-GHz Intel® Xeon® E5-2690 platform.

While NVDIMM devices have been a mainstay of embedded storage use cases, for example, as a write cache for storage controllers or a fast log device for a larger storage engine, they have not enjoyed wide deployment in general purpose storage applications. Small capacities and the logistics of battery management relegate current generation NVDIMMs to niche use cases. Intel memory modules based on 3D XPoint technology promise to remove the obstacles to general purpose use cases. However, before the capacity and latency advantages of persistent memory can be fully realized, significant changes to an operating system's traditional storage stack are required. This is a direct parallel to the changes already introduced for low latency storage. The goal this time is to enable persistent memory to commit data to media with just one to two instructions. Linux has already implemented DAX, and further improvements in the development pipeline will allow applications to avoid operating system overheads in the IO path to a persistent memory device [40].

Removing or amortizing the overhead of recurring coordination with a file system to commit updates to DAX-capable files is a topic of ongoing development. A mechanism that completely removes the file system from the equation is Device-DAX. Device-DAX turns an entire volume's worth of persistent memory, a capacity range that would typically house an entire file system, into a device-special file that can be DAX mapped. This gives an application full ownership of the memory and no requirement to notify the operating system kernel or file system of new updates. However, that level of raw access and responsibility is only suitable for a small class of purpose-built applications. Work continues to bring the bypass capabilities of Device-DAX to a DAX file system-based file. We believe that other major operating systems will provide similar enabling.

Leading applications may quickly take advantage of these capabilities while it may be some time before other data center applications are prepared to consume the full capabilities. Subsector-sized byte-aligned reads, and writes that can be committed in a few instructions change the traditional assumptions of storage. Researchers have already shown significant advantage. Oukid *et al.* [41] rewrote an in-memory database to be persistent memory aware. For their application they showed that the application-measurable performance of persistent memory comes close to that of DRAM. They further exposed the benefit of significantly shorter downtime in the event of power failure. By incorporating persistent memory awareness in applications, other researchers [42], [43] have exposed significant performance advantages for use cases ranging from key–value stores, sorts and joins, and transactional logging. Operating systems will continue to mature their persistent memory access mechanisms enabling a new generation of data-center applications.

VII. CONCLUSION

Ultrafast Intel Optane SSDs based on a new technology, 3D XPoint memory, have arrived to the market place. These SSDs deliver data with a latency rivaling that of the system itself, upending our decades-old view of storage as the slowest system component. Deep changes to the system making way for this advance have already taken place, including moving SSDs to PCIe, eliminating the HBA, and tuning of operating system code paths and policies. With these changes in place, the performance offered by these new SSDs is available to the system.

Low latency storage accesses blur the line between storage and memory. SSD queue-depth-based measurements no longer make sense, because much of the IOs lifetime is spent in the system not the storage device. SSDs should be measured like memory with bandwidth (IOPS) versus latency measurements. Paging to storage, long avoided, is now a viable strategy. When a 4-kB page is only 16 μs away, many applications can use storage like an additional level of memory through existing operating system paging. So these new high-performance SSDs are not just measured like memory, but may also be used like memory.

Application-based studies show that the performance of Intel Optane SSDs is accessible by applications. RocksDB, an important key–value store, shows a 3× improvement when run on an Intel Optane SSD. The Real Time Analytics ACT benchmark also shows greater than a 3× performance advantage when run on an Intel Optane SSD. In both cases the lower latency, and more ideal performance under load (better QoS) are responsible for these system level performance advantages. Moreover, the door is now open for solution innovation to make use of the ability to achieve high throughput and deliver excellent QoS, something not possible with NAND-based SSDs.

When used as a paging store, the Intel Optane SSD again results in similar application performance but enables another key advantage—an increased data set size. This effect was shown for both a memcached variant called fatcache and for an in-memory analytics application. For these examples, the SSD is fast, the system is fast, and the application benefits in runtime decrease or data set size increase.

While important, fast SSDs are not the endpoint for this technology. Operating systems are already undergoing the changes need to expose the new memory technology as persistent memory with the creation of DAX mode. Applications will benefit both for traditional file system uses with persistent memory file systems, and through direct use of persistent memory for maximum benefit. The advent of new fast memories creates an exciting time in systems architecture with opportunities to exploit fast SSDs in the near term for immediate system level application performance benefits on the way to even greater benefits with persistent memory. ■

Acknowledgment

This work is the result of many, especially Intel's SSD team, of which the authors are part. The authors would like to thank especially A. Fazio for leading the way and J. Smits, W. Fang, S. Vyas, R. Medel, S. Mehta, K. Putnam, J. Tang, and A. Torok for many of the experiments and measurements quoted in this paper. Finally, they would like to thank their customers who generously shared their workloads with them. Intel, Intel Core, Intel Xeon, Intel Optane, and 3D XPoint are trademarks of Intel Corporation or its subsidiaries in the U.S. and/or other countries. Other names and brands may be claimed as the property of others.

REFERENCES

[1] (May 17, 2016). *IBM Scientists Achieve Storage Memory Breakthrough*. [Online]. Available: https://www-03.ibm.com/press/us/en/pressrelease/49746.wss

[2] C. Nguyen, D. Burkard, K. Dobbins, and C. Bohac (Aug. 5, 2016). *MRAM Improvements to Automotive Non-Volatile Memory Storage*. [Online]. Available: https://www.everspin.com/file/1101/download

[3] A. Shilov. (Aug. 12, 2016). *Western Digital to Use 3D ReRAM as Storage Class Memory for Special-Purpose SSDs*. [Online]. Available: http://www.anandtech.com/show/10562/western-digital-to-use-3d-reram-as-storage-class-memory-for-specialpurpose-ssds

[4] Crucial. *Crucial NVDIMMs: Powerful and Persistent Server Memory Performance*. [Online]. Available: http://www.crucial.com/usa/en/memory-server-nvdimm

[5] V. Prabhakaran, T. L. Rodeheffer, and L. Zhou, "Transactional flash," in *Proc. 8th USENIX Conf. Oper. Syst. Des. Implement. (OSDI)*, San Diego, CA, USA, 2008.

[6] J. Condit *et al.*, "Better I/O through byte-addressable, persistent memory," in *Proc. 22nd ACM Symp. Oper. Syst. Principles (SOSP)*, Big Sky, MT, USA, 2009, pp. 133–146.

[7] H. Volos, A. J. Tack, and M. M. Swift, "Mnemosyne: Lightweight persistent memory," in *Proc. 15th ACM Int. Conf. Archit. Support Program. Lang. Oper. Syst. (ASPLOS)*, Newport Beach, CA, USA, 2011, pp. 91–104.

[8] C. Hellwig, R. Zwisler, and B. Harrosh (Apr. 1, 2015). *[PATCH 2/2] pmem: Add a Driver for Persistent Memory*. [Online]. Available: https://lwn.net/Articles/640114/

[9] X. Wu and A. L. N. Reddy, "SCMFS: A file system for storage class memory," in *Proc. Int. Conf. High Perform. Comput., Netw., Storage Anal. (SC)*, Seattle, WA, USA, 2011, pp. 1–11.

[10] R. Zwisler, V. Verma, S. Kumar, M. Wilcox, and R. Gittins *Persistent Memory File System*. [Online]. Available: https://github.com/linux-pmfs/pmfs

[11] S. R. Dulloor *et al.*, "System software for persistent memory," in *Proc. 9th Eur. Conf. Comput. Syst. (EuroSys)*, Amsterdam, The Netherlands, 2014, p. 15.

[12] A. Kudryavtsev. (Feb. 1, 2016). *SSD as a System Memory? Yes, With ScaleMP's Technology*. [Online]. Available: https://storagebuilders.intel.com/blog/ssd-as-a-system-memory-yes-with-scalemps-technology-2

[13] A. Rudoff, "Programming models for emerging non-volatile memory technologies," *Login*, vol. 38, no. 3, p. 39–45, 2013.

[14] J. Corbet. (Apr. 15, 2015). *Persistent Memory Support Progress*. [Online]. Available: https://lwn.net/Articles/640113/

[15] A. Rudoff. *NVM Library*. [Online]. Available: http://pmem.io/nvml/

[16] A. P. Foong, B. Veal, and F. T. Hady, "Towards SSD-ready enterprise platforms," in *Proc. 1st Int. Workshop Accel. Data Manage. Syst. Using Modern Processor Storage Archit. (ADMS)*, Singapore, 2010, pp. 15–21.

[17] J. L. Hennessy and D. A. Patterson, *Computer Architecture: A Quantitative Approach*, 5th ed. San Francisco, CA, USA: Morgan Kaufmann, 2011.

[18] Z. Kerekes. (2007). *SSD Market History*. [Online]. Available: http://www.storagesearch.com/ssd-history-2007.html

[19] NVM Express. [Online]. Available: http://nvmexpress.org/

[20] M. Bjøling, J. Axboe, D. Nellans, and P. Bonnet, "Linux block IO: introducing multi-queue SSD access on multi-core systems," in *Proc. 6th ACM Int. Syst. Storage Conf. (SYSSTOR)*, Haifa, Israel, 2013, p. 22.

[21] J. Yang, D. B. Minturn, and F. Hady, "When poll is better than interrupt," in *Proc. 10th USENIX Conf. File Storage Technol. (FAST)*, San Jose, CA, USA, 2012, p. 3.

[22] C. Mason. (Mar. 8, 2017). *Btrfs*. [Online]. Available: https://btrfs.wiki.kernel.org/index.php/Main_Page

[23] Oracle. (2010). *Oracle Solaris ZFS Administration Guide*. [Online]. Available: http://docs.oracle.com/cd/E19253-01/819-5461/

[24] P. Sehgal, S. Basu, K. Srinivasan, and K. Voruganti, "An empirical study of file systems on NVM," in *Proc. 31st IEEE Symp. Mass Storage Syst. Technol. (MSST)*, Santa Clara, CA, USA, May/Jun. 2015, pp. 1–14.

[25] J. Xu and S. Swanson, "NOVA: A log-structured file system for hybrid volatile/non-volatile main memories," in *Proc. 14th USENIX Conf. File Storage Technol.*, Santa Clara, CA, USA, 2016, pp. 323–338.

[26] J. Yang. (Dec. 23, 2016). *Pmbench*. [Online]. Available: https://bitbucket.org/jisooy/pmbench

[27] T. Chen. (Oct. 20, 2016). *mm/swap: Regular Page Swap Optimizations*. [Online]. Available: https://lwn.net/Articles/704359/

[28] J. Axboe. (Mar. 13, 2017). *fio HOWTO*. [Online]. Available: https://github.com/axboe/fio/blob/master/HOWTO

[29] Open Source Development Lab. *Iometer*. [Online]. Available: http://iometer.org/

[30] C. Ramseyer. (Mar. 14, 2015). *How We Test HDDs and SSDs*. [Online]. Available: http://www.tomshardware.com/reviews/how-we-test-storage.4058.html

[31] K. Grimsrud, "IOPS schmIOPS! What really matters in SSD performance," in *Proc. Flash Memory Summit*, Santa Clara, CA, USA, 2013.

[32] B. Jacob, *The Memory System: You Can't Avoid It, You Can't Ignore It, You Can't Fake It*. San Rafael, CA, USA: Morgan and Claypool, 2009.

[33] D. Borthakur. (Nov. 21, 2013). *Under the Hood: Building and open-Sourcing RocksDB*. [Online]. Available: https://www.facebook.com/notes/facebook-engineering/under-the-hood-building-and-open-sourcing-rocksdb/10151822347683920/

[34] Aerospike. (Apr. 12, 2016). *White Paper: Building an Enterprise-Grade Database Architecture for Mission-Critical, Real-Time Applications*. [Online]. Available: http://www.aerospike.com/resource/white-paper-building-an-enterprise-grade-database-architecture-for-mission-critical-real-time-applications/

[35] B. Bulkowski, "NVMe, storage class memory and operational databases: Real-world results," in *Proc. Flash Memory Summit*, Santa Clara, CA, USA, 2016.

[36] Aerospike. *Aerospike Certification Tool (ACT)*. [Online]. Available: http://www.aerospike.com/act-for-ssds/

[37] J. Gray and P. Shenoy, "Rules of thumb in data engineering," in *Proc. 16th IEEE Int. Conf. Data Eng. (IDCE)*, San Diego, CA, USA, Mar. 2000, pp. 3–10.

[38] B. Fitzpatrick. *Memcached*. [Online]. Available: http://memcached.org/

[39] M. Rajshekhar and Y. Yue (2013). *fatcache*. [Online]. Available: https://github.com/twitter/fatcache

[40] D. Williams and T. Kasanicky, "Managing persistent memory," in *Proc. Vault Linux Storage Filesyst. Conf. (Vault)*, Raleigh, NC, USA, 2016.

[41] I. Oukid, D. Booss, W. Lehner, P. Bumbulis, and T. Willhalm, "SOFORT: A hybrid SCM-DRAM storage engine for fast data recovery," in *Proc. 10th Int. Workshop Data Manage. New Hardw. (DaMoN)*, Snowbird, MT, USA, 2014, p. 8.

[42] S. D. Viglas, "Write-limited sorts and joins for persistent memory," *Proc. VLDB Endowment*, vol. 7, no. 5, pp. 413–424, 2014.

[43] J. Huang, K. Schwan, and M. K. Qureshi, "NVRAM-aware logging in transaction systems," *Proc. VLDB Endowment*, vol. 8, no. 4, pp. 389–400, 2014.

ABOUT THE AUTHORS

Frank T. Hady received the B.S. and M.S. degrees in electrical engineering from the University of Virginia, Charlottesville, VA, USA and the Ph.D. degree in electrical engineering from the University of Maryland, College Park, MD, USA.

He is an Intel Fellow and Director of the Storage Technology Group within the Non-Volatile Memory Solutions Group (NSG), Intel Corporation, Hillsboro, OR, USA. His team researches future SSD and systems level storage advances, specifies the architectures defining Intel's SSDs and other storage products, and partners to create industry standards. He was previously the chief architect of 3D XPoint Storage in NSG, leading architecture definition of Intel Optane SSDs and systems advances for fast storage. He has authored or coauthored more than 30 published papers on topics related to networking, storage, and I/O innovation, and he holds more than 30 U.S. patents.

Annie Foong received the Ph.D. degree in computer engineering from the University of Wisconsin, Madison, WI, USA.

She jointed Intel Corporation, Hillsboro, OR, USA, in 1999, where she is currently a Principal Engineer with the NVM Solutions Group. She is the lead architect for Intel Optane Data Center SSDs.

Bryan Veal received the M.S. degree in computer science from the University of Georgia, Athens, GA, USA, in 2005.

He joined Intel Corporation, Hillsboro, OR, USA, in 2005, where he is currently a Platform Architect for Intel Optane Data Center SSDs.

Dan Williams joined Intel Corporation, Hillsboro, OR, USA, in 2002, where he is currently a Software Engineer with the Open Source Technology Center. He is a Linux kernel developer, enabling next-generation storage technologies and persistent memory. He is a member of the Linux Foundation Technical Advisory Board.