Exploring Performance Characteristics of the Optane 3D Xpoint Storage Technology

JINFENG YANG, BINGZHE LI, and DAVID J. LILJA, University of Minnesota, Twin Cities

Intel's Optane solid-state nonvolatile storage device is constructed using their new 3D Xpoint technology. Although it is claimed that this technology can deliver substantial performance improvements compared to NAND-based storage systems, its performance characteristics have not been well studied. In this study, intensive experiments and measurements have been carried out to extract the intrinsic performance characteristics of the Optane SSD, including the basic I/O performance behavior, advanced interleaving technology, performance consistency under a highly intensive I/O workload, influence of unaligned request size, elimination of write-driven garbage collection, read disturb issues, and tail latency problem. The performance is compared to that of a conventional NAND SSD to indicate the performance difference of the Optane SSD in each scenario. In addition, by using TPC-H, a read-intensive benchmark, a database system's performance has been studied on our target storage devices to quantify the potential benefits of the Optane SSD to a real application. Finally, the performance impact of hybrid Optane and NAND SSD storage systems on a database application has been investigated.

CCS Concepts: • **General and reference** → **Evaluation**; **Performance**; • **Information systems** → **Storage class memory**; Hierarchical storage management;

Additional Key Words and Phrases: 3D Xpoint memory, nonvolatile memory, solid-state drive, Optane SSD, NAND SSD, performance evaluation, relational database management system

ACM Reference format:
Jinfeng Yang, Bingzhe Li, and David J. Lilja. 2020. Exploring Performance Characteristics of the Optane 3D Xpoint Storage Technology. *ACM Trans. Model. Perform. Eval. Comput. Syst.* 5, 1, Article 4 (February 2020), 28 pages.
https://doi.org/10.1145/3372783

1 INTRODUCTION

Solid-state storage drives (SSDs) based on NAND flash technology can deliver several orders of magnitude better performance than conventional hard-disk drives (HDDs). Most notably, the NAND SSD can directly access the physical location of the data without mechanical arm movement. This emerging innovation not only saves the seek time that usually existed within

This work was supported in part by the Center for Research in Intelligent Storage (CRIS), which is supported by National Science Foundation grant no. IIP-1439622 and member companies. Any opinions, findings, and conclusions or recommendations expressed in this material are those of the authors and do not necessarily reflect the views of the NSF.
Authors' addresses: J. Yang, B. Li, and D. J. Lilja, Department of Electrical and Computer Engineering, University of Minnesota, 200 Union Street SE, Minneapolis, MN 55455, USA; emails: {yang3116, lixx1743, lilja}@umn.edu.

2376-3639/2020/02-ART4 $15.00
https://doi.org/10.1145/3372783

conventional rotating disks but also reduces power consumption. Because of these performance advantages, the NAND SSD has rapidly become one of the most prevalent persistent storage devices during the last few decades and has been widely discussed in both industry and academia [8, 11, 13, 32, 33, 39, 49].

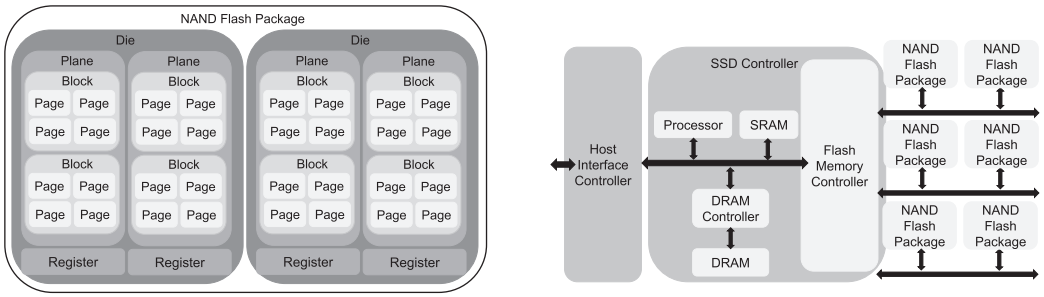
Although the NAND flash-based SSD has achieved significant performance improvements, the huge performance gap between high-performance processors and slow storage devices has not changed. In addition, NAND SSD performance will degrade dramatically under some specific situations. For instance, the random read latency can be 10× slower than sequential read due to the prefetching failures of the internal read-ahead mechanism; the write performance degrades significantly when the request size is not aligned on a multiple of the page size [13]; and the update operation requires longer completion time because of garbage collection issues. To pursue fast and stable performance, numerous new technologies have been developed, including Phase Change Memory (PCM) [47], Conductive Bridging RAM (CBRAM) [30], Ferroelectric RAM (FeRAM) [41], and others.

Intel recently released 3D Xpoint technology as one of these emerging nonvolatile technologies and integrated it into the solid-state driver to replace conventional NAND flash cells. This newly developed storage device is known as Optane SSD [7] and can deliver fast I/O access latency and high throughput. Most notably, 3D Xpoint memory can support functionalities that NAND flash cannot, such as byte addressability and in-place update. Due to these performance advantages, the Optane SSD is expected to be a good candidate to accelerate the slower storage subsystem in the future. However, few research studies have been presented to discuss the unique characteristics of the Optane SSD. Previous publications [18, 21] and product specifications [6] either focused mainly on the potential performance impacts of Optane SSD to real applications or gave limited information about its intrinsic characteristics.

In this study, by designing various types of I/O workloads, intensive experiments were conducted on the Optane SSD, and the collected results were analyzed. The purpose of this article is to provide an understanding of the unique performance behavior of 3D Xpoint storage technology rather than to carry out a performance competition with the conventional NAND SSD. This article makes the following contributions:

(1) Several basic performance behaviors of the Optane SSD, including I/O access latency, internal parallelism, and performance consistency under highly intensive I/O workload, have been evaluated.
(2) Based on multiple custom-designed microbenchmarks and methodologies, further details of the intrinsic characteristics of the Optane SSD have been explored, including byte addressability, the elimination of write-driven garbage collection, the read disturb issue, and the tail latency problem.
(3) The implications of the Optane SSD for a real database system were studied using the TPC-H benchmark. By analysis of the collected results, it was noticed that Optane SSD delivers asymmetric performance accelerations to different types of queries. The potential reasons for these results were explored.
(4) A database application was implemented on a hybrid Optane and a NAND SSD storage system and its performance evaluated. Several key factors that can impact this hybrid system's performance significantly were pointed out.

The rest of this article is organized as follows: Section 2 introduces background information. Section 3 describes the experimental environment and tools. Section 4 presents the custom-designed microbenchmarks and the analysis of the collected results. Section 5 describes a study of the performance of a database application on both Optane and NAND SSDs and an investigation



(a) NAND flash package (b) SSD internal architecture

Fig. 1. SSD internals.

of the implications of the Optane SSD for a real application. Section 6 investigates the key factors that could impact database query performance for a hybrid Optane and NAND SSD storage system. Section 7 discusses related work, and Section 8 concludes the article.

2 BACKGROUND

2.1 NAND Flash Memory

NAND flash memory is a type of memory medium that is widely used in solid-state drives. As described in Figure 1(a), a *NAND flash package* typically consists of a number of *dies* (or chips). Each die is composed of one or more *planes* and a number of *registers* for buffering I/O. A single plane contains multiple *blocks*. A block is the smallest *erase* unit inside NAND flash memory. Each block consists of 128 or 256 *pages*. A page is the smallest unit on which normal *read* and *write* operations can be performed. A typical page contains a 4KB (or larger) data area and a 128-byte metadata area for error correction coding (ECC) [8].

NAND flash memory has some unique features. First, the read-write (program) latency delivered by NAND flash memory is not uniform. The write operation typically takes a longer time than the read operation. Second, due to the asymmetric unit size between program and erase operations, a log-structure-based out-of-place update and a garbage collection mechanism are required for write operations. Finally, a NAND flash memory has a limited number of program/erase (P/E) cycles, usually 1,000 to 100,000 cycles [11, 29], and cannot work once it wears out. Hence, manufacturers usually reserve an extra space (called overprovisioning) to replace bad memory cells and improve the lifespan of a NAND SSD.

2.2 3D XPoint Memory

3D Xpoint is one of the new types of nonvolatile memory media recently released by Intel. It has been integrated for the first time into a solid-state drive, the Optane SSD, to replace conventional NAND flash memory. Compared with NAND flash, 3D Xpoint memory exhibits different characteristics from two main aspects. First, 3D XPoint memory is byte addressable, in contrast to NAND flash that accesses data in a page unit. By benefiting from this byte addressability, an Optane SSD internal controller can break a 4KB I/O into several smaller data chunks and spread them across multiple channels concurrently to hide single-I/O latency, thus boosting performance. Second, 3D Xpoint memory supports the update-in-place operation, which means that the Optane SSD internal controller does not need to perform extra read, write, and erase operations, also known as read-modify-write, when overwrite requests are submitted.

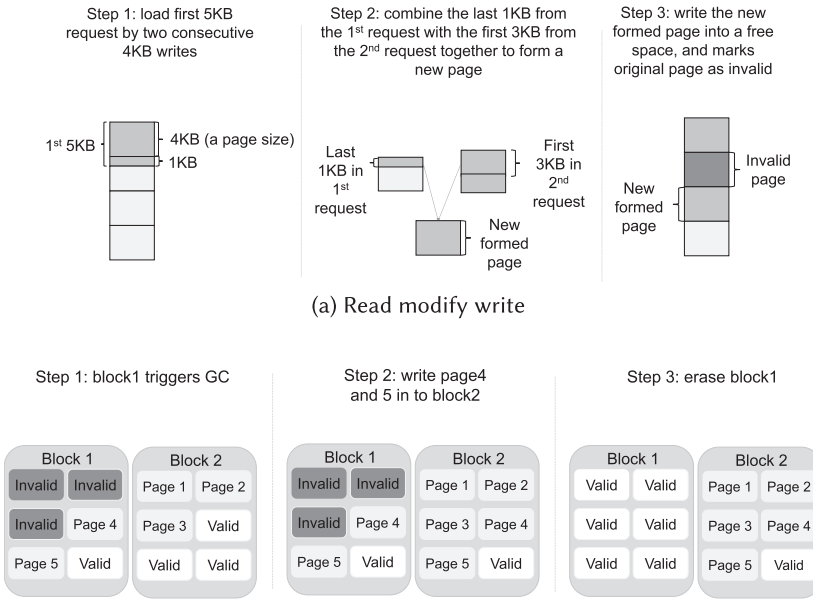


Fig. 2. NAND SSD write issues.

2.3 SSD Internal Architecture

As shown in Figure 1(b), a conventional SSD is composed of several main components, including a host interface, an internal controller, an embedded RAM, and persistent memory media (usually NAND flash memory). The host interface establishes a physical connection between the storage device and the host. It transfers logic commands and data from/to the host through a predefined protocol (e.g., SCSI, SATA, or PCIe). The SSD controller manages data placements in NAND flash memory. To improve the SSD's lifetime, the internal controller is also assigned to process wear leveling and garbage collection mechanisms [8]. To deliver low I/O access latency, a high-end NAND SSD is usually equipped with an embedded RAM to buffer data temporarily for read and write operations [17].

Moreover, to improve bandwidth, SSD manufacturers usually spread flash memory cells across multiple independent channels, packages, dies, and planes to exploit SSD internal parallelism [20, 43]. By using this interleaving technique, the SSD controller can fetch/load multiple pages from/into different cells simultaneously and consequentially obtain high bandwidth.

2.4 NAND SSD Write Issues

Read-Modify-Write. As discussed before, a page is the smallest unit on which normal read and write operations can be performed within a NAND SSD. To maintain high performance, any requests submitted to a NAND SSD are required to align on a page size. However, in real applications, the request size may not be bounded to a page size. This scenario is defined as a subpage request [22] and results in fragmentation and endurance problems. In addition, because NAND flash memory does not allow data update-in-place, a subpage write typically causes a NAND SSD to trigger more internal operations than necessary. As described in Figure 2(a), when a subpage request is submitted, the rest of the page (to which the subpage belongs) is read into embedded RAM and

combined with the subpage request to form a full page. Then the SSD controller writes this newly formed page into a free space. Meanwhile, the old page is marked as invalid and will be recycled during garbage collection. Finally, the flash translation layer [24–26, 34] updates the mapping table. The processes just described are known as *read-modify-write* [8]. Because read-modify-write usually requires additional read, write, and update mapping table operations, it can significantly impact foreground write performance.

Write-Driven Garbage Collection [13]. Unlike a hard disk drive, a NAND SSD does not allow data update-in-place. When an update operation is submitted, instead of overwriting, the SSD controller writes the updated data into a free space. Meanwhile, the pages containing the old information are marked as invalid. As the number of invalid pages continues to increase, the NAND SSD may no longer have enough free space to store incoming data. As a result, the internal controller triggers the garbage collection mechanism to recycle invalid pages. During garbage collection, as described in Figure 2(b), to prevent data loss, the valid pages within the target block must first be read out and written into a free space. The SSD controller then erases all pages within the target block to release more free space. Garbage collection typically degrades NAND SSD performance tremendously due to the extra internal operations (reads, writes, and erases) and the long latency of the erase operation ($\sim 2ms$) [13].

2.5 Read Disturb Error

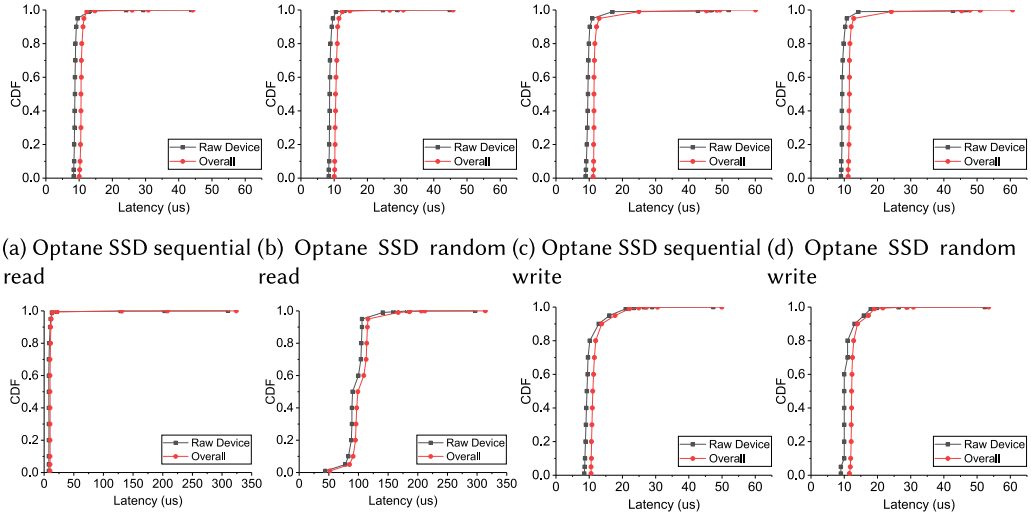
Read disturb is an intrinsic error within many types of memory media, such as NAND flash memory [15] and Phase Change Memory (PCM) [31, 40]. When a high read count is required on a given page, the analog voltages of neighboring cells could be disturbed enough to change their stored information. To prevent data loss, NAND SSD manufacturers usually set a threshold on the number of read cycles. Once the number of reads on a given page/block exceeds the threshold, the internal controller forces a refresh to copy the neighboring pages into a new location. Meanwhile, the controller marks the old pages as invalid and recycles them during garbage collection. Because the read disturb issue triggers internal data migration and garbage collection with certain probabilities, it is one of the factors that limits NAND SSD performance.

3 EXPERIMENTAL SETUP

Experiments were conducted on a Dell Precision Tower 3620 workstation equipped with four Intel core i7-6700 3.4GHz processors and 16GB DDR4 DIMMs. The Ubuntu 16.04.4 LTS operating system and Ext4 file system were installed on the workstation. Two types of SSDs were directly attached on-board through a PCIe x16 NVMe 30 x4 interface, including an Intel 168GB DC P4800X **Optane** SSD [6] and an Intel 2TB DC P3700 NAND SSD [4]. The NAND SSD was used as a baseline device.

For these experiments, the Completeness Fairness Queuing scheduler (the default I/O scheduler for the Linux kernel) was changed to the Noop scheduler. The Noop scheduler [28] implements the simplest first-in-first-out (FIFO) algorithm, leaving I/O performance optimization to the storage device. For fast storage devices such as SSD, Noop can outperform other I/O schedulers in most cases [11]. In addition, all read and write requests were performed by direct I/O to bypass the buffer cache within the file system.

To investigate the intrinsic characteristics of the Optane SSD, the Flexible I/O tester (FIO) [2] was used as a benchmarking tool. In the experiments, various I/O workloads were generated by FIO with different parameters, such as I/O type, request size, queue depth, access rate, and others. During the experiment, each workload was run three times to observe typical behavior. For random read and write requests, a random number generator was configured inside the FIO to guarantee that the read and write order was not repeated on different runs.



(a) Optane SSD sequential read (b) Optane SSD random read (c) Optane SSD sequential write (d) Optane SSD random write

(e) NAND SSD sequential read (f) NAND SSD random read (g) NAND SSD sequential write (h) NAND SSD random write

Fig. 3. Basic I/O performance behaviors.

4 PERFORMANCE STUDIES

The experiments described in this section investigated the unique features of the Optane SSD from multiple aspects, including basic I/O performance behaviors, advanced internal parallelism, performance consistency under highly intensive I/O workload, the influence of unaligned request size, the elimination of write-driven garbage collection, the read performance variation potentially resulting from the read disturb issue, and the tail latency problem. The detailed experimental methodologies and results analysis are discussed below.

4.1 Basic I/O Performance Behaviors

Because it uses high-performance 3D Xpoint memory media, the Optane SSD is expected to deliver high performance. The experiments described in this subsection examine the basic read/write latency of the Optane SSD under different access patterns, using the NAND SSD as a baseline device for comparison.

The experiment involved four workloads, including *sequential read*, *random read*, *sequential write*, and *random write*, with a 4KB request size and a queue depth of 1. Two timers were invoked from two different start points to record the latency: one started recording when the I/O request was generated to measure overall I/O latency; the other started recording when the I/O request was sent into the NVMe submission queue [45] to measure raw device latency. Both timers were stopped once the completion interrupt was received. For each workload, 1 million I/O requests were submitted into storage, and their I/O latency distribution was recorded.

Figure 3 plots the cumulative distribution functions (CDFs) of the I/O access latency for four different workloads on both Optane and NAND SSDs. The results reveal that the NAND SSD delivered nearly the same overall I/O latency as the Optane SSD for the *sequential read*, *sequential write*, and *random write* workloads ($\sim 10 - 14us$). This high performance of the NAND SSD can be attributed to its read-ahead mechanism and the potential of the internal buffer. For sequential read, the internal controller prefetches needed pages early into the internal buffer and sends them to the host when required. For both sequential and random write operations, the internal controller caches pages

into the NAND SSD internal buffer and sends the completion interrupt to the host immediately. The cached pages are loaded later in the background into the proper location inside the NAND flash cells. Because the pages are fetched/loaded from/into the NAND SSD internal buffer through a PCIe interface, the NAND SSD delivered similar I/O latency for *sequential read* (~10*us*) and *write* (~11*us*) operations. This assumption can be confirmed from Figures 3(e), 3(g), and 3(h). However, due to mistakes in prefetching, a tiny portion of the data is read from the NAND flash cells. The NAND SSD has a very wide I/O latency distribution range on sequential read, as shown in Figure 3(e). In addition, the performance enhancement provided by the read-ahead mechanism is highly correlated with the access pattern. As shown in Figure 3(f), under the random access pattern, because of the low prefetching accuracy of the read-ahead mechanism, NAND SSD random read overall latency increased to 100*us*, which amounts to a 10X degradation of the sequential read pattern.

In contrast, the Optane SSD delivered uniform I/O latency regardless of access pattern (either sequential or random). This result confirms that the Optane SSD does not need a read buffer to boost its performance, especially for sequential read. It is also worth noting that the Optane SSD has nearly the same I/O access latency between read (~10.72*us*) and write (~11.95*us*, a little bit lagged behind read) operations. In addition, compared with the NAND SSD, the Optane SSD provides a more compact I/O latency distribution under the read access pattern. As shown in Figures 3(a) and 3(b), the I/O latency of the Optane SSD was concentrated in the 10*us* to 45*us* range for both sequential and random read. On the contrary, the NAND SSD exhibited a very wide I/O latency distribution, from 10*us* to 324*us* for sequential read, and from 52*us* to 314*us* for random read. This wide range of the I/O latency distribution for both Optane and NAND SSDs is known as tail latency, which will be discussed in more detail later in this article. Finally, by comparing overall and raw device latency on the Optane SSD, it can be observed that the application level introduces 20% (2*us*) latency while processing I/Os. This could be an important factor limiting Optane SSD performance, which hopefully can be fixed by further optimization.

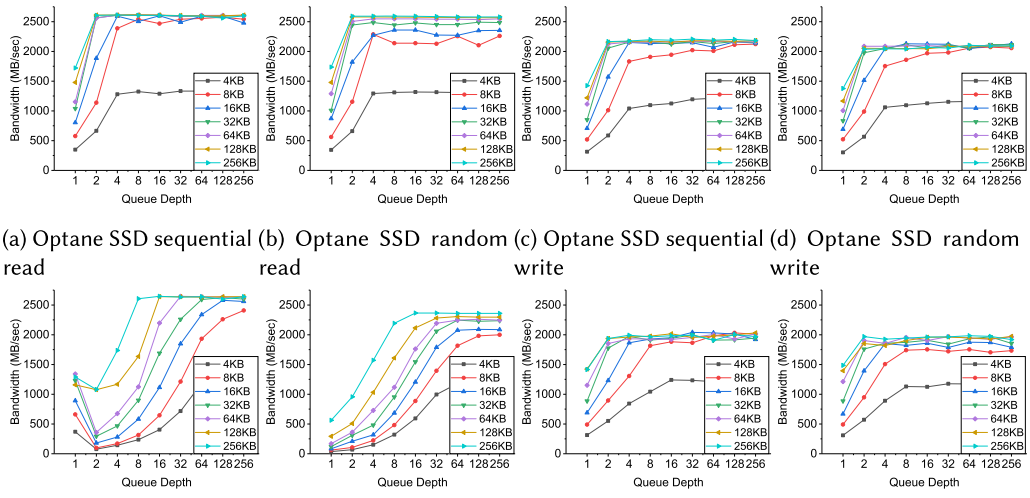
The Optane SSD's advanced performance can be attributed to two factors. First, the 3D Xpoint memory delivers higher performance than NAND flash. Second, to hide single-I/O access latency, the Optane SSD spreads a single 4KB I/O over multiple 3D Xpoint memories across different channels [18]. In contrast, the NAND SSD accesses a single NAND flash cell to satisfy a 4KB (or larger) I/O. Due to these two improvements, the Optane SSD delivers a faster and more compact I/O latency distribution than the NAND SSD.

4.2 Advanced Interleaving Technique

To provide high bandwidth, NAND SSD manufacturers normally use an interleaving technique to access pages from multiple channels in parallel. The Optane SSD inherits a similar channel-based interleaving technique to that of the NAND SSD, but with advanced implementation [18]. This subsection quantifies the potential benefits of this advanced design inside the Optane SSD.

In this experiment, four different I/O workloads, including sequential read, random read, sequential write, and random write, were developed. For each workload, the queue depth was varied from 1 to 256, and their corresponding bandwidth was recorded with different request sizes from 4KB to 256KB.

Figure 4 shows the results of this experiment. Compared with the Optane SSD's unprecedented performance enhancement, the bandwidth gains delivered by the interleaving technique inside the NAND SSD were limited. Even though the NAND SSD has a maximum bandwidth close to that of the Optane SSD (2.5GB/s for read and 2GB/s for write), it required a much greater queue depth than the Optane SSD to deliver this performance. As shown in Figures 4(e) and 4(f), under the 4KB request size, the NAND SSD could provide its maximum sequential and random read bandwidth (1.2GB/s) only when its queue depth was enlarged to 128. In contrast, the Optane SSD delivered the



(a) Optane SSD sequential read (b) Optane SSD random read (c) Optane SSD sequential write (d) Optane SSD random write

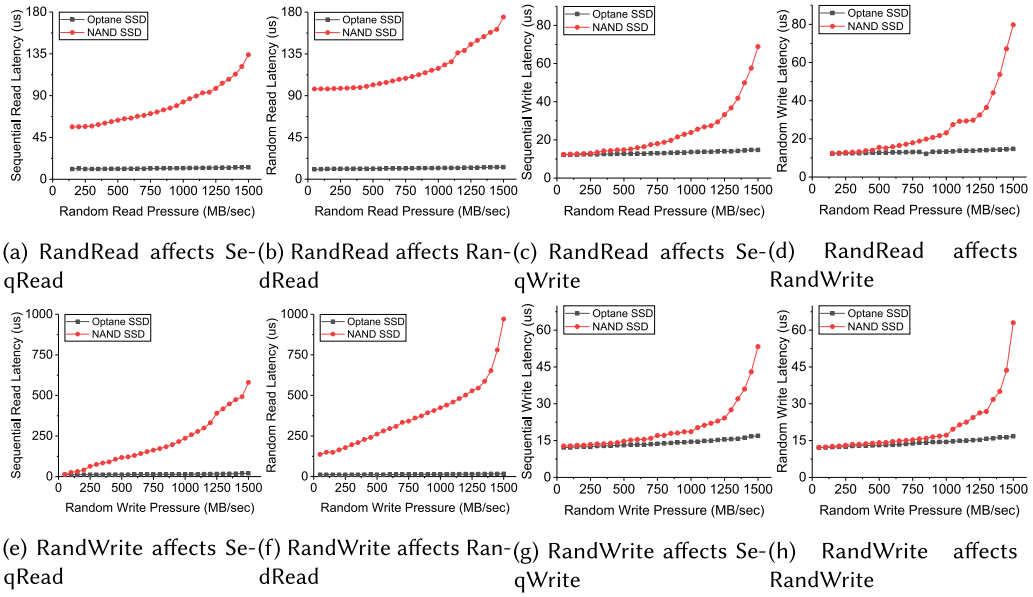
(e) NAND SSD sequential read (f) NAND SSD random read (g) NAND SSD sequential write (h) NAND SSD random write

Fig. 4. Interleaving technique.

same bandwidth with a queue depth of only 4, which was 32X smaller than the NAND SSD. Similar results were also observed on both sequential and random write operations. With the same queue depth, even with an internal buffer, the NAND SSD still could not provide the same bandwidth improvement as the Optane SSD for smaller request sizes such as 4KB and 8KB. Moreover, the interleaving technique used within the NAND SSD is easily affected by other processes. As shown in Figure 4(e), when the queue depth was decreased to 2, the NAND SSD sequential read bandwidth degraded tremendously and recovered only when the queue depth was continuously increased to be larger than 16. This performance degradation issue resulted from interference between the internal parallelism and the read-ahead mechanisms inside the NAND SSD [12].

Unlike the NAND SSD, which spreads numerous pages (4KB or larger) across multiple NAND flash channels in parallel to deliver higher bandwidth, the interleaving technique inside the Optane SSD works in a more efficient manner. Within the Optane SSD, the internal controller distributes a single 4KB I/O across multiple independent channels concurrently [18]. This means that the I/O size accessed from a single 3D Xpoint memory medium could be smaller than a normal page size. The Optane SSD can do this because the 3D Xpoint is a byte-addressable memory medium and can support access to small data chunks (smaller than 4KB). With the benefit of this advanced interleaving technique and high-performance 3D Xpoint memory, the Optane SSD achieves its maximum bandwidth with a much smaller queue depth than the NAND SSD. As illustrated in Figures 4(a) to 4(d), for smaller request sizes (4KB, 8KB, and 16KB), the Optane SSD bandwidth starts to become saturated after the queue depth is increased to 4. For larger request sizes (equal to or larger than 32KB), the Optane SSD can reach its maximum bandwidth with a queue depth of only 2. This is possible because the larger request size has partially triggered the Optane SSD's internal parallelism, and therefore it requires less queue depth to provide its maximum bandwidth.

In addition, the results obtained here show that the Optane SSD provides similar performance gains for different I/O access patterns (either sequential or random) as queue depth increases. This occurs because the 3D Xpoint memory has nearly the same access latency for both sequential and random access patterns (Section 4.1). Moreover, because the Optane SSD directly accesses I/Os from the host to the storage media through a hardware-only path [18], the performance



(a) RandRead affects SeqRead (b) RandRead affects RandRead (c) RandRead affects SeqWrite (d) RandRead affects RandWrite

(e) RandWrite affects SeqRead (f) RandWrite affects RandRead (g) RandWrite affects SeqWrite (h) RandWrite affects RandWrite

Fig. 5. Performance consistency under highly intensive I/O workload.

degradation problem that exists inside the NAND SSD for the sequential read access pattern was not observed with the Optane SSD.

Finally, for both Optane and NAND SSDs, the maximum bandwidth delivered by 4KB and larger request sizes (8KB or larger) was found to have a 2X difference. This may have occurred because the *clustered page* [29] size in both Optane and NAND SSDs is 8KB, where the clustered page size is defined by the number of integrated pages to use the SSD interleaving technique.

4.3 Performance Consistency under Highly Intensive I/O Workload Pressure

The previous subsections have explored the characteristics of the Optane SSD using a single-I/O process. However, in real applications, multiple processes (or threads) may submit their I/O requests into storage simultaneously. Under this situation, the performance of a storage device can be impacted significantly due to internal resource competition. To understand the performance consistency issues of the Optane SSD under highly intensive I/O workloads, intensive experiments and measurements were carried out and are described in this subsection.

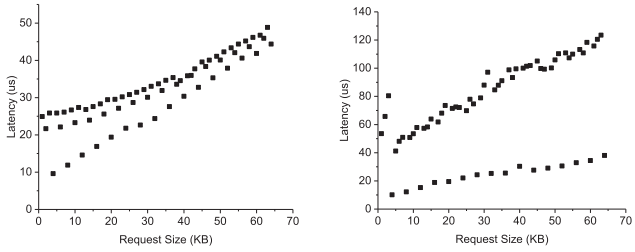
In this experiment, two types of *background* I/O-intensive workloads were generated by FIO, including random read and random write. The random I/O access pattern was used because it triggers more internal operations than sequential patterns and can be expected to reflect the worst performance of storage devices. By varying the data access rate of the background I/O workload in terms of how many data points are read/written from/into the storage device per unit time, the background I/O workload pressure can be controlled. To show how storage performance is affected, when the above background I/O workload was running, a *foreground* process was called to submit a million I/O requests to storage with a 4KB request size and a queue depth of 1, and its average I/O access latency was recorded. Figure 5 plots the experimental results, where the x-axis represents I/O access pressure. For instance, if the random read pressure is equal to 400MB/s, then the background I/O workload accesses data from the storage device at a rate of 400MB/s. The y-axis records the I/O latency in microseconds for four different access patterns.

As discussed in Section 4.1, with the benefit of its internal buffer and read-ahead mechanism, under a single-I/O process, the NAND SSD can deliver nearly the same I/O access latency as the Optane SSD for sequential read, sequential write, and random write. However, as the background I/O workload pressure was continuously increased, the potential benefits of the NAND SSD's internal optimization mechanisms were greatly impacted. As a result, NAND SSD performance degraded tremendously. As illustrated in Figures 5(a), 5(c), and 5(d), after the background random read pressure was increased to 1,500MB/s, the NAND SSD's I/O access latency for sequential read, sequential write, and random write degraded by 13.25X, 5.78X, and 6.19X, respectively, compared with the results for a single-I/O process. Poorer results were observed under the random write I/O workload. As shown in Figures 5(e) to (f), when the background random write pressure was increased to 1,500MB/s, the foreground sequential and random read latency increased to 580us and 971us, respectively. Finally, from Figure 5(a), it can be observed that when the random read I/O pressure was set to 50MB/s, the prefetching accuracy of the NAND SSD's read-ahead mechanism was significantly affected, and its corresponding sequential read latency increased to 56.32us, amounting to a 5.6X degradation compared with the results for a single-I/O process (10.11us). This NAND SSD performance degradation issue can be attributed to resource competition among its internal optimization mechanisms and the physical nature of the NAND flash memory. For instance, under background random read pressure, as the data access rate is continuously increased, the NAND SSD controller can more easily trigger internal refresh to migrate data into a new location to prevent read disturb errors. As a result, the performance of the foreground process can be tremendously affected. For the background random write I/O workload, because it may result in internal garbage collection, the performance of the foreground process was further degraded.

In contrast with the NAND SSD's inconsistent performance behavior, the experimental results revealed that the Optane SSD maintained low I/O access latency even under high-pressure I/O workload. Analyzing the results showed that under heavy random read I/O pressure (up to 1,500MB/s), the I/O access latency of the Optane SSD on sequential read, random read, sequential write, and random write increased by only 2.2us, 2.39us, 2.62us, and 2.65us, respectively, compared with results for the single-I/O process. Similar results were also observed for the random write I/O pressure case. When the random write pressure was increased to 1,500MB/s, the I/O latency of the Optane SSD for sequential read, random read, sequential write, and random write increased by 8.23us, 6.26us, 4.77us, and 4.6us, respectively. It is apparent that the background write pressure had a stronger impact on foreground process performance than the read pressure. This may have occurred because the Optane SSD's write operation requires more resources (either hardware or software resources) than the read operation. This consistent performance behavior of the Optane SSD can be attributed to the following factors. First, the Optane SSD is equipped with a well-optimized internal controller [18], which enables the Optane SSD to schedule I/O processes more efficiently than the NAND SSD. Second, because I/O requests are performed directly from the host to the storage media by a hardware-only path [18], the extra latency that results from the NAND SSD's internal resource competition under high-pressure I/O workload is removed from the Optane SSD. Third, because an update-in-place-enabled 3D Xpoint memory is used, the extra resource consumption resulting from the NAND SSD's internal garbage collection is eliminated from the Optane SSD. Due to these factors, the Optane SSD can deliver relatively stable performance under increasing background I/O workload pressure.

4.4 Performance Influence of Unaligned Request Size

The NAND SSD performs normal read and write operations based on the page unit. To sustain high performance, any requests submitted to the NAND SSD are required to be aligned on a page size. However, in real applications, an I/O request may no longer be bound to a page size. Especially



(a) Optane SSD (b) NAND SSD

Fig. 6. Performance influence of different request sizes.

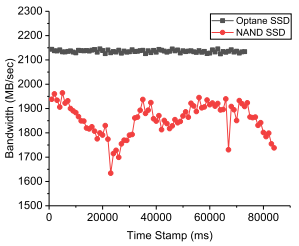


Fig. 7. Write-driven garbage collection.

under the sequential write access pattern, a request with unaligned size leads the NAND SSD to trigger more internal operations than necessary, a sequence known as read-modify-write [8], and therefore degrades NAND SSD performance. However, if the NAND flash is replaced by a byte-addressable 3D Xpoint memory, does the Optane SSD still need to perform I/O based on a certain chunk size to avoid performance degradation?

To answer this question, the authors developed an FIO-based I/O microbenchmark to record the sequential write latency with different request sizes. This benchmark was similar to [29], but with certain modifications. In this experiment, to avoid interference by the host file system, only the raw device performance was recorded. To avoid resource competition between multiple I/O jobs, only one process ran the workload with a queue depth of 1. To improve reliability, 1 million sequential write requests were submitted into storage, and their average I/O access latency was then calculated.

The same benchmark was run on both Optane and NAND SSDs by varying the request size from 1 to 64KB with granularity 1KB; their corresponding average sequential write latencies are plotted in Figure 6. From Figure 6(b), a dramatic average write latency increment can be observed from the NAND SSD when the request size is not a multiple of 4KB, which is similar to the result presented in [29]. This performance degradation issue resulted from the NAND SSD's extra read, write, and update mapping table operations during the read-modify-write process (Section 2.4). In addition to this, a new phenomenon was observed. For I/Os with a request size smaller than a page size (4KB), their corresponding I/O access latency for the 1KB, 2KB, and 3KB request sizes increased tremendously to 54us, 66us, and 80us, respectively. This could have been caused by multiple read-modify-write processes occurring within a single page. For instance, a 1KB request triggers four read-modify-write processes to package four consecutive requests into one page. Furthermore, this small unaligned request size also makes the number of invalid pages within NAND flash memory increase rapidly. As a result, the NAND SSD controller is more likely to invoke garbage collection to recycle these invalid pages.

3D Xpoint storage technology has two notable differences from NAND flash memory. First, because 3D Xpoint memory supports data update-in-place, the extra read, write, and update mapping table operations caused by read-modify-write are removed. Second, because 3D Xpoint is a byte-addressable memory medium, it can access small data chunks (smaller than a 4KB page inside the NAND SSD) directly. Because of these advantages, within the Optane SSD, the performance degradation resulting from unaligned request size is reduced significantly compared with the NAND SSD results. However, as shown in Figure 6(a), a performance degradation in the Optane SSD can still be observed when the request size is not a multiple of 4KB. There are two ways to explain these Optane SSD performance degradation issues. First, although 3D Xpoint memory is byte addressable, because it is integrated into a block storage device, the I/Os submitted to the

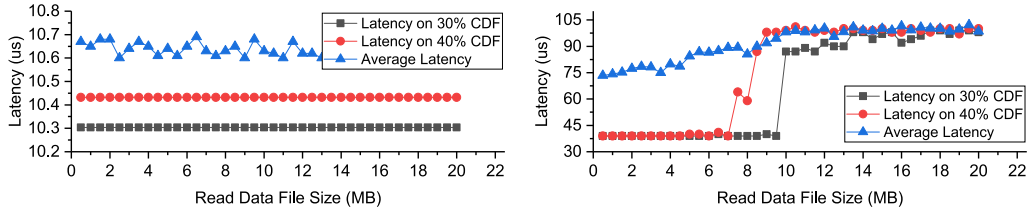
Optane SSD should conform to the block device's functionality—block-based access—to mitigate performance degradation. Second, the Optane SSD optimizes its internal I/Os by spreading a single 4KB across multiple memory cells [18]. Any request that is not a multiple of 4KB cannot be well optimized and therefore has longer I/O access latency. In addition, the results for the Optane SSD show that there are three latency layers. For the lowest layer, because all I/O requests are aligned on a page size and can be well optimized, they have the lowest access latency. For the middle layer, because the I/O sizes are even values and can be partially optimized, their latency is between best and worst. For the highest layer, because the I/O request sizes are odd numbers that are hard to optimize, the Optane SSD delivers its highest access latency.

Finally, unlike the NAND SSD's results that the average sequential write latency increases proportionally to the unaligned request size and always maintains a large gap when the request size is a multiple of 4KB, the performance degradation of the Optane SSD that results from the unaligned request size is reduced as the request size increases. When the request size is larger than 50KB, the Optane SSD delivers almost the same write latency whether or not the request size is bounded on 4KB. The authors believe that this occurs because as the request size increases, more internal parallelisms are invoked within the Optane SSD, which hides the single sequential write latency.

4.5 Performance Degradation of Write-Driven Garbage Collection

As discussed in Section 2.4, the NAND SSD does not allow data update-in-place. When the content within a page is modified, instead of overwriting, the NAND SSD controller writes the updated information into a new page and marks the original page as invalid. As the number of invalid pages continues to increase, the SSD controller invokes internal garbage collection to recycle invalid pages to release more available space. The entire process is known as write-driven garbage collection [13]. Because garbage collection typically triggers extra read, write, and erase operations in the background, it degrades a foreground process's performance once it happens. Moreover, because NAND flash memory has a limited number of program/erase cycles, typically 10,000 to 100,000 [11], invoking garbage collection frequently leads a NAND SSD to reach its lifespan early.

However, by replacing the NAND flash memory inside the solid-state drive with the 3D XPoint memory, the Optane SSD can support data update-in-place. Benefiting from this technology innovation, the Optane SSD is believed to have removed write-driven garbage collection and is expected to deliver more consistent performance under the write access pattern than the NAND SSD. This subsection explores the potential of enabling data update-in-place through a custom-designed microbenchmark. Before the test, both Optane and NAND SSDs were filled with random writes, leaving only 150GB free space. Then a 150GB file was randomly written to fill the remaining available space inside the storage with a carefully selected request size and queue depth to enable both Optane and NAND SSDs to deliver their maximum bandwidth. In this experiment, the request size and queue depth were set to 16KB and 256, respectively. After preprocessing, both the Optane and the NAND SSDs were full. At the end, the previous 150GB file was overwritten following the exact random write orders, request size, and queue depth used during the preprocessing stage, and the corresponding bandwidth was recorded into a log file. To avoid interference, this log file was preserved on the host memory at the beginning and was flushed into the storage until the overwrite process was terminated. Because the storage device was already filled before the workload was run, the NAND SSD had to trigger garbage collection during overwriting to release space to store the updated information. In addition, because manufacturers usually reserve a specific space inside the NAND SSD (called overprovisioning) to store valid pages during garbage collection, there is enough available space to guarantee that the NAND SSD can complete garbage collection during overwriting. This experiment used random write because it more easily triggers garbage collection than sequential write.



(a) Optane SSD (b) NAND SSD

Fig. 8. Internal buffer size.

The microbenchmark was run on both Optane and NAND SSDs, and the corresponding bandwidths were plotted in Figure 7. As the results show, when overwrite requests were submitted into a full storage device, the random write performance of the NAND SSD was impacted substantially due to background garbage collection. During the entire process, a sharp bandwidth drop was observed for the NAND SSD, and the corresponding performance degradation was 19% (from the maximum 2GB/s to the minimum 1.62GB/s). Although NAND SSD performance can be recovered after garbage collection, its maximum bandwidth can only be maintained within a very short time interval and drops immediately when the next garbage collection starts. On the contrary, because 3D Xpoint memory allows data update-in-place, the performance degradation issues that result from write-driven garbage collection are removed from the Optane SSD. During the entire process, the Optane SSD maintained consistent bandwidth, around 2.1GB/s. Even though the Optane SSD could not provide exactly the same bandwidth during overwriting, the bandwidth difference was only 0.5% (from the maximum 2.14GB/s to the minimum 2.13GB/s), which can be attributed to system variation.

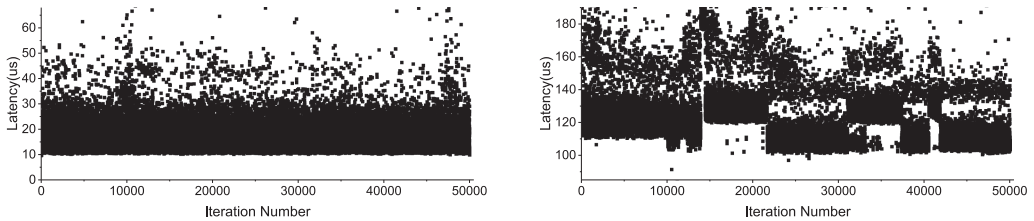
4.6 Read Disturb Issue

This subsection describes the investigation of whether the Optane SSD faces the same read disturb issue as the NAND SSD and how much performance degradation the Optane SSD experiences during an internal forced refresh. In this experiment, the target page was accessed with high read count on both the Optane and NAND SSDs, and their corresponding access latencies were recorded. To avoid caching the target page into the storage device's internal buffer, the random read option was used, and the read data size was set to 2X larger than the estimated internal buffer size. The capacity of the internal buffer can be roughly estimated as follows: randomly read a data file repeatedly (3X) with 4KB request size and record the I/O access latency of each request. When the data file size is smaller than the read buffer size, repeatedly accessing the same data file would allow the internal buffer to cache a portion of the frequently accessed pages. As a result, the cached pages would have much smaller access latency than those that were not cached, and the average read latency of a single 4KB page is relatively small. As the data file size continues to increase and becomes larger than the internal buffer capacity, because the cached pages must be evicted from the internal buffer at each turn and all pages must be read from storage media, all requests have similar read latency, and the average latency becomes high. To see clearly how read latency varies as read data file size is continually increased, the I/O access latency was plotted on the 30%, 40% cumulative density function (CDF) level. The average latency is shown in Figure 8. Figure 8(b) shows a sharp read latency increment on the 30%, 40% CDF level, whereas the read data file size increases to 8MB and 10MB, respectively. And the average read latency of a single-I/O request becomes constant after the data file size exceeds 10MB. Based on these results, the internal buffer size of a NAND SSD can be roughly estimated as 10MB. However, read latency variations were

ALGORITHM 1: Read Disturb Test

Input: F, /*target file stored on storage device with size 21MB*/

```
1: procedure TESTING PROCEDURE
2:     for i ← 0 to 50,000 do
3:         clean host memory
4:         lseek(F, 0, set) /*read at the beginning of the file*/
5:         timer start
6:         read_file(F, 4KB) /*read target page*/
7:         timer end
8:         read_file(F, 20MB) /*evict target page from SSD internal buffer by random read an extra 20MB
   file*/
9:         record target page read latency into a logfile
10:    Flush logfile to storage device
```



(a) Optane SSD (b) NAND SSD

Fig. 9. Read disturb issue.

not observed from the Optane SSD. This result confirms that the Optane SSD directly accesses I/O from the host to the storage media without using firmware optimization techniques such as a internal buffer.

After the estimated read buffer size had been determined, the target page (4KB, the page size has already been explored in Section 4.4) was accessed repeatedly on both Optane and NAND SSDs to test the read disturb issue. The microbenchmark was described in Algorithm 1. Some implementation details must be mentioned here: first, to avoid caching the target page in the host, DRAM was clean hosted before the read; second, the target page was read by direct I/O and the read latency recorded into a log file; third, to avoid interference, the log file was preserved in host DRAM and flushed into storage until the program terminated; fourth, after the target page was read, an extra 20MB of data were immediately random read to evict the target page from the SSD internal buffer and guarantee that in each iteration, the target page would be read from the storage media; and finally, the number of read cycles on the target page was set to a large value (50,000) to trigger SSD internal forced refresh to prevent the read disturb error.

Figure 9 shows a plot of the experimental results. The read latency variation from both Optane and NAND SSDs can be observed. The observed results may arise from internal forced refresh to prevent the read disturb error on both 3D Xpoint and NAND flash memory. In addition, from Figure 9(b), there are clear latency jumps at iteration numbers 14700, 32000, and 40000. This may have occurred because the NAND SSD triggered internal garbage collection to recycle invalid pages. As a result, it degraded the foreground read performance further. The NAND SSD's random read latency immediately dropped when garbage collection was completed at iteration numbers

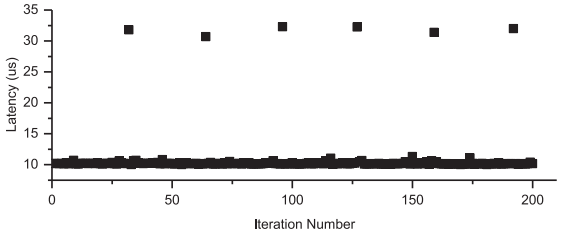


Fig. 10. Optane SSD, only access the target page repeatedly.

22000, 37500, and 41000. In contrast, because garbage collection was eliminated from the Optane SSD, the clear latency jumps and drops that existed for the NAND SSD were not observed.

However, from Figure 9, it is hard to determine the precise number of read cycles that triggers the read disturb issue and how much performance would degrade during the SSD internal forced refresh. This difficulty is compounded because, besides accessing the target page repeatedly, it is also necessary to random read an extra 20MB data file in each iteration to evict the target page from the NAND SSD internal buffer, which also causes the read disturb issue and leads to an SSD internal forced refresh. To answer these questions, the target page was simply accessed repeatedly from the Optane SSD; the corresponding results are plotted in Figure 10. In this experiment, plotting started after the latest high read latency occurred that was potentially due to the read disturb issue. These results reveal a periodic latency increment at approximately every 30 iterations and a corresponding performance penalty resulting from the internal forced refresh of 20us.

4.7 Tail Latency

As data sizes continue to grow, storage performance is becoming critical to many data-intensive applications today. To deliver low and stable response time, modern applications demand that 99.9% of all requests must be answered within a limited time interval [46]. However, due to the impact of garbage collection and other background mechanisms, the NAND SSD typically has a very wide I/O latency distribution, known as tail latency. This performance instability introduces milliseconds of delay and leads to a 1.5X to 2.0X slowdown of storage systems [19]. In this subsection, the tail latency of both Optane and NAND SSDs is explored under different scenarios (high-pressure I/O workloads, garbage collection, and read disturb). To reflect how background processes impact tail latency, the results in Section 4.1 were used as a *baseline*, and it was assumed that only one process submitted random read requests to storage. In this experiment, the random read request was used to examine tail latency. This choice was made because most modern NAND SSDs are equipped with an internal buffer (Section 4.1). Using other I/O requests (such as sequential read, sequential write, and random write) cannot reflect the real performance of NAND flash memory.

To understand how background I/O workload impacts storage performance stability, the experiment described in Section 4.3 was repeated. A background process was called to submit random read requests to storage with a certain access rate (*200 MB/s*, *400 MB/s*, and *800 MB/s*). Another foreground process was then invoked to submit a million random read requests into storage, and its I/O latency distribution was recorded on the 99th, 99.5th, 99.9th, 99.95th, and 99.99th percentiles, as shown in Table 1. To indicate the I/O latency distribution range, the I/O latency on the 50th percentile, called the median latency, was also recorded. The 50th percentile latency can be thought of as representing the average latency of all I/O requests. The results showed that as background I/O workload pressure was continuously increased, the NAND SSD's I/O latency became distributed over a wider range. For instance, when the background I/O workload pressure was increased to 800MB/s, the NAND SSD's foreground random read latency ranged from 101us to 832us for the

Table 1. Tail Latency Comparison

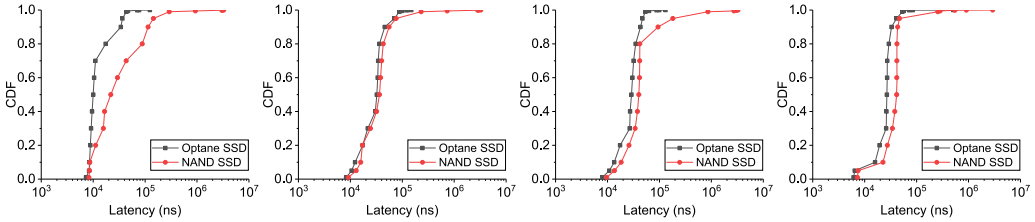
Percentile	50	90	99.5	99.9	99.95	99.99
Optane SSD (*us*)						
Baseline	8.6	10.6	13.1	24.7	29.1	44.8
200MB/s Pressure	8.9	14.5	16.5	26.0	29.3	45.2
400MB/s Pressure	9.0	22.4	23.7	30.0	31.4	49.3
800MB/s Pressure	9.3	17.5	19.8	29.6	33.0	54.0
Garbage Collection	9.5	33.5	41.2	45.9	47.9	56.0
Read Disturb	8.6	12.6	13.4	24.9	28.3	45.7
NAND SSD (*us*)						
Baseline	90	141	159	182	186	297
200MB/s Pressure	92	148	162	179	189	318
400MB/s Pressure	97	151	167	190	208	344
800MB/s Pressure	101	212	253	343	449	832
Garbage Collection	100	2,474	2,737	2,989	3,064	3,228
Read Disturb	100	285	293	314	408	2,089

50th to the 99.99th percentile. In contrast, the Optane SSD's foreground random read latency varied only from 9.3*us* to 54*us* for the 50th to the 99.99th percentile, or a 16.4X smaller range than that of the NAND SSD.

Garbage collection is another factor that contributes long tail latency to the NAND SSD. To investigate this, the experiment described in Section 4.5 was repeated. This experiment overwrote a 150GB file to storage by a background I/O process, then submitted a foreground random read request to storage and recorded its latency distribution. To minimize the impact of resource competition due to background I/O workload, for the overwrite operation, the request size was decreased to 4KB and the queue depth to 1. As shown in Table 1, because of background garbage collection, the NAND SSD's 99.99th percentile latency was increased to 3,228*us*, which is 32X greater than its corresponding median latency. On the contrary, benefiting from the elimination of garbage collection, the Optane SSD had only a 5.9X difference between the 50th and 99.99th percentile latency.

In addition to the background I/O workload process and garbage collection, data refresh can also intensify internal resource competition in a storage device, resulting in longer tail latency. The experiment described in Section 4.6 was therefore repeated. A 20MB data file was randomly read 50,000 times and its I/O latency distribution recorded. As shown in Table 1, due to background data refresh and garbage collection (Section 4.6), the NAND SSD produced a very wide I/O latency distribution, and its 99.99th percentile latency reached 2,080*us*, which was 20.9X larger than its median latency. In contrast, the impact of the Optane SSD's internal data refresh on foreground performance was negligible. As shown by the results, even though the Optane SSD triggered internal data refresh, it delivered almost the same I/O latency distribution as the baseline. This result also indicates that Optane SSD's tail latency under the single-I/O process potentially resulted from its internal data refresh to avoid read disturb error.

By summarizing the results from the different scenarios, it can be concluded that the background I/O process, internal data refresh, and garbage collection are the three main factors that contribute long tail latency to the NAND SSD. In contrast, the Optane SSD achieved a similar response trend among the various scenarios. In other words, the negative impact of internal data refresh and overwriting on the Optane SSD's tail latency was reduced, but the background I/O process becomes a main factor that can contribute to long tail latency.



(a) prn_1 (b) proj_0 (c) usr_2 (d) web_2

Fig. 11. MSR trace implementation.

Table 2. MSR Trace Configuration [42]

	Number of IOs (Millions)		Total Request Size (GB)	
	Write	Read	Write	Read
prn_1	2.77	8.46	30.78	181.35
proj_0	3.70	0.53	144.27	8.97
usr_2	1.99	8.58	26.47	415.28
web_2	0.04	5.14	0.78	262.82

4.8 Real Trace Implementation

To explore the impact of the Optane SSD on a real storage subsystem, as described in this subsection, an MSR trace [42] was implemented on both the Optane and NAND SSDs and the I/O latency distribution of each request recorded, as shown in Figure 11. The trace characteristics are summarized in Table 2.

As Figure 11 shows, on the same percentile level, the Optane SSD delivered faster I/O access latency than the NAND SSD. Meanwhile, as discussed in Section 4.7, due to the elimination of garbage collection and low performance degradation during internal data refresh, the Optane SSD produced a more compact I/O latency distribution than the NAND SSD. For instance, for proj_0 trace, the 99.99th percentile latency of the NAND SSD increased to 3,256.4*us*. In contrast, the latency of the Optane SSD on the same percentile was only 149.7*us*, which is 22X lower than the NAND SSD.

5 IMPLICATIONS FOR A REAL DATABASE SYSTEM

Database systems are one of the prime I/O-intensive applications. Their performance is typically limited by laggard storage subsystems. However, by incorporating a new type of nonvolatile 3D Xpoint memory, the Optane SSD is expected to provide a new insight for these I/O-intensive applications. This section quantifies the potential benefits of an Optane SSD in a relational database management system (RDBMS) using the TPC-H, a read-intensive benchmark [1]. Using the TPC-H benchmark, Yang and Lilja [50] explored the limiting factors that introduce extra I/O cost into database systems, including read amplification issues, high buffer pool cache miss rates, and inefficient use of temporary tables. However, this discussion lacked a detailed analysis of TPC-H queries from the storage level. This article mainly focuses on evaluating Optane SSD performance on an RDBMS with a detailed explanation. The experimental results in this study have shown that Optane SSD delivers up to 6.5X speedup for the TPC-H benchmark.

5.1 TPC-H Benchmark and RDBMS

The TPC-H is one of the online analytical processing benchmarks that is widely used in both academia and industry. This benchmark program is composed of a number of business-oriented

Table 3. Database Tables' Information of TPC-H Benchmark

Table Name	Abbreviation of Each Table's Name	Table Size
SUPPLIER	S	121MB
REGION	R	96KB
PARTSUPP	PS	9.1GB
PART	P	2.1GB
ORDERS	O	14GB
NATION	N	112KB
LINEITEM	L	62GB
CUSTOMER	C	1.8GB

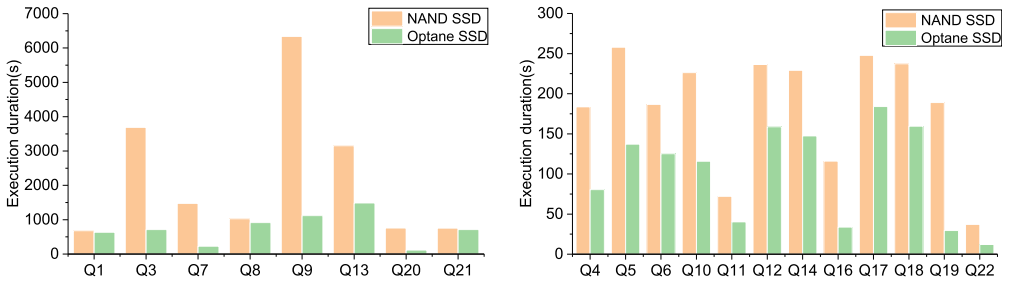


Fig. 12. TPC-H queries Result. X-axis represents query name.

queries and data modifications. The queries and database tables are carefully designed to have broad industry relevance. The performance reported by the TPC-H benchmark reflects multiple capacities of the tested system.

To evaluate the potential of the Optane SSD in a real I/O-intensive application, the TPC-H schema was used in this study to derive an RDBMS (MySQL) with a scale factor of 60. After all the database tables had been loaded into storage, the database size increased to 89.1GB. The size of each table is listed in Table 3. The initial setup on MySQL [5] is described below.

(1) Database Storage Engine: We use Innodb as the default storage engine of MySQL. Compared with others, such as MyISAM, Innodb provides more functionality, including commit, roll-back, and crash recovery.
(2) Buffer Pool Size: The buffer pool size is set to 4GB for caching database tables and indices in memory.
(3) The Number of Innodb I/O Threads: The number of background threads used to perform I/O requests is set to the Innodb default value of 4.

5.2 TPC-H Performance Evaluation

In the experiment described in this section, the TPC-H queries were run on both Optane and NAND SSDs. Their execution durations are shown in Figure 12. Here, we do not include Q2's and Q15's results since those two queries failed to complete during testing for unknown reasons. From these results, the Optane SSD achieved more than 3X higher performance than the NAND SSD for queries Q3, Q7, Q9, Q13, Q16, Q19, Q20, and Q22. For the rest of the queries, the Optane SSD

Table 4. TPC-H Access Pattern

	Q1	Q3	Q4	Q5	Q6	Q7	Q8	Q9	Q10	Q11
seq read	99%	6%	96%	91%	99%	5%	89%	5%	93%	66%
rand read	1%	94%	4%	9%	1%	95%	11%	95%	7%	34%
seq write	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%
	Q12	Q13	Q14	Q16	Q17	Q18	Q19	Q20	Q21	Q22
seq read	99%	4%	96%	20%	24%	99%	13%	8%	99%	49%
rand read	1%	96%	4%	76%	76%	1%	87%	92%	1%	51%
seq write	0%	0%	0%	4%	0%	0%	0%	0%	0%	0%

Note, there is no random write for all queries.

Table 5. TPC-H Repeat Access Rate

	Q1	Q3	Q4	Q5	Q6	Q7	Q8	Q9	Q10	Q11
RA rate	1.0	4.6	1.0	1.0	1.0	2.2	1.1	6.2	1.0	1.0
	Q12	Q13	Q14	Q16	Q17	Q18	Q19	Q20	Q21	Q22
RA rate	1.0	16.8	1.0	1.0	1.0	1.0	1.0	1.4	1.0	1.0

delivered 1.1X to 3X speedup compared to the NAND SSD. To understand the underlying reasons for these results, the block-level I/O requests of the TPC-H queries were captured by blktrace [9] and analyzed.

The first step was to study the access patterns of each query, including sequential read, random read, sequential write, and random write. We say a page was sequentially accessed if it was read by a database read-ahead mechanism. Otherwise, it was defined as random access. A page was the smallest data unit of MySQL, and the page size was 16KB. As shown in Table 4, all TPC-H queries were read intensive. However, this observation is still not adequate to explain some queries' extremely long execution duration, such as Q9 with 6,339*s* total execution duration in the NAND SSD. To find the reason for this, *the block-level repeat access rate* of each query was tested, where the block-level repeat access rate was defined to be the total number of accessed pages from the block level divided by the number of distinct pages accessed from the block level during querying. Both numbers can be counted from each query's block-level trace. For instance, during a query, the block-level accessed page sequence is p1, p3, p4, p3, p1, p4, p5, p3. Then the block-level repeat access rate is computed by the number of total accessed pages, 8, divided by the number of distinct accessed pages, 4 (p1, p3, p4, and p5), which equals 2. Table 5 presents the results. In the remainder of the article, we call the block-level repeat access rate the repeat access rate for simplicity.

Summarizing the results from Tables 4 and 5 reveals that all TPC-H queries can be separated into three categories: (1) A query is randomly and repeatedly read. Here, a query is considered to be randomly read if its random read percentage exceeds 50%. In the same way, a query is considered to be repeatedly accessed if its repeat access rate is greater than 1.1. These queries are Q3, Q7, Q9, Q13, and Q20. (2) A query is considered to be randomly but not repeatedly read, including Q16, Q17, Q19, and Q22. (3) A query is sequentially but not repeatedly read, including Q1, Q4, Q5, Q6, Q8, Q10, Q11, Q12, Q14, Q18, and Q21. By combining the results in Figure 12, it is apparent that for the first category of queries that are randomly and repeatedly accessed, processing them on a NAND SSD requires extremely long execution times on the order of thousands of seconds. There are several explanations for this result. First, the NAND SSD has around 10X lower random read performance than the Optane SSD (Section 4.1) because randomly accessing the database files involves a huge amount of I/O time. Second, in the experiments performed in this study, four

independent threads performed I/O requests concurrently, as discussed in Section 4.3, and this highly intensive I/O workload degraded NAND SSD performance tremendously because of the resource competition of its internal optimization firmware. Finally, as described in Section 4.6, accessing a given page repeatedly degrades read performance due to the read disturb issue. In addition, because this may also trigger NAND SSD internal garbage collection, foreground read performance is impacted further. For these reasons, executing the queries in category 1 on a NAND SSD results in extremely long I/O time, limiting overall database performance. In contrast, because the Optane SSD benefits from a number of advanced technologies, including fast random read speed, absence of garbage collection, and low performance degradation during internal data refresh, processing these same queries on an Optane SSD showed more than 3X performance speedup. Especially for Q7, the performance acceleration delivered by Optane SSD can be as much as 6.5.

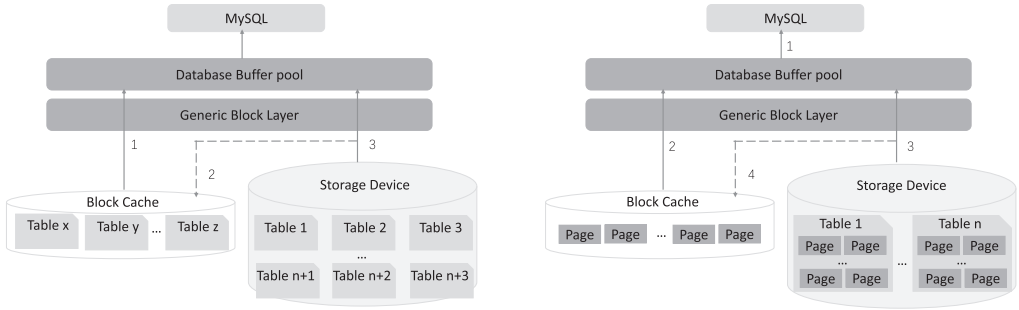
For the second category's queries that are randomly but not repeatedly read, processing them on the NAND SSD also leads to long execution duration. However, because most pages are accessed only once during querying and because the NAND SSD has a low probability of triggering internal data refresh and garbage collection, the execution times for these queries, on the order of hundreds of seconds, are still acceptable compared with thousands of seconds of execution time for the first query category. In contrast, due to the order-of-magnitude better performance of the Optane SSD on random read access patterns, executing these queries on an Optane SSD provided more than 3X performance acceleration. However, Q17 was an exception; its performance was improved only 1.3X by the Optane SSD. To find the reason for this, the I/O, 61.2*s*, and computation time, 186.22*s*, of Q17 were both recorded on the NAND SSD. The results indicated that Q17 was a computation-intensive query. Even though processing Q17 on the Optane SSD achieved 5.7X speedup on I/O time, 10.8*s*, because the computation time, 183.95*s*, was still dominant, the overall performance speedup was limited.

Finally, for the third category's queries that were sequentially but not repeatedly read, processing them on the Optane SSD achieved only a 1.1X to 3X performance speedup. There are two explanations for these results. First, because the performance of these queries was mainly limited by the host computation time instead of the I/O latency [50], the I/O time speedup provided by Optane SSD was weakened. Second, because only four I/O threads were invoked during the query and the NAND SSD's internal read mechanism could still schedule I/O jobs efficiently, the NAND SSD delivered similar, although slightly lower sequential read performance than the Optane SSD. As a result, the Optane SSD achieved limited performance acceleration.

6 INVESTIGATIONS OF THE HYBRID STORAGE SYSTEM FOR A DATABASE APPLICATION

Although the Optane SSD delivers substantial performance improvement, its price is much higher than that of conventional storage devices. In the real world, only a few Optane SSDs may be available, which cannot be used to store entire database tables, but many low-price NAND SSDs may be available. In this circumstance, designing a hybrid storage system may become a feasible approach to use existing storage devices efficiently. In this hybrid storage system, the fast storage device, Optane SSD, is used as a block cache to buffer highly valuable tables or pages temporarily, and the slow storage device, conventional NAND SSD, is used as a persistent storage to keep all database tables.

Designing a hybrid system is not a new topic. A previous study [37] proposed a table placement algorithm for a hybrid NAND SSD and HDD storage architecture, as shown in Figure 13(a). It assumed that a NAND SSD has much faster random read performance than an HDD, but that both provide the same performance for sequential read. By using information extracted from the database query execution plan tree, the above algorithm tends first to place tables that will be



(a) *Hybrid system with table placement algorithm.* Step 1: While processing a database query, the block cache is first consulted. If the target tables have been cached, read them from the block cache. Step 2: Otherwise, block the query, migrate target tables into the block cache, and resume the query. Step 3: If there is no free space on the block cache to place target tables, read target tables from the storage device.

(b) *Hybrid system with caching algorithm.* Step 1: While processing a database query, if the required page has been cached in the host, read it from the database buffer pool. Step 2: Otherwise, read the required page from the block cache. Step 3: If the required page is still not found, fetch it from the storage device. Step 4: Meanwhile, the required page has also been a cache candidate to be buffered into the block cache.

Fig. 13. Hybrid storage system.

accessed by index into the NAND SSD and then to process the query. However, the performance improvement of this algorithm is highly dependent on the prediction accuracy of the query's execution plan tree. For instance, a table that is predicted to be accessed by index may be sequentially scanned in a real application. Canim et al. [10] proposed a temperature-aware caching (TAC) algorithm for a hybrid NAND SSD and HDD system; its architecture is as presented in Figure 13(b). During database querying, through measuring the temperature of each region, TAC only admits the pages from the hot region. However, since it is possible that a rarely accessed page is from a hot region, TAC suffers miss-caching issues.

Although previous work has some drawbacks, they do provide some insights that prompt the designer to think about hybrid storage architecture. This section describes a performance investigation of a hybrid Optane and NAND SSD storage system for a database application with a hybrid table replacement algorithm and hybrid caching algorithm, respectively. Proposing a proper algorithm is not the goal of this article; instead, the focus is on finding and evaluating the key factors that could impact the performance of this hybrid storage system and thereby provide some suggestions to researchers or engineers to help them design a high-performance hybrid storage system.

6.1 Hybrid System with Table Placement Algorithm

This subsection investigates the performance impacts of the table placement algorithm for the hybrid Optane and NAND SSD system. We will address the following three questions: (1) How does the hybrid table placement algorithm affect the performance of the three types of query that were mentioned in Section 5, including sequentially read queries, randomly read queries, and randomly and repeatedly read queries? (2) Is it always true that placing a larger table or using more Optane SSD space can deliver higher performance for database queries? (3) What are the key factors that could impact a hybrid table placement algorithm's performance for database queries?

To answer these questions, one representative query was chosen from each query type, including Q10, a sequentially read query; Q16, a randomly read query; and Q3, a randomly and repeatedly read query. The clause of each query is shown in Figure 14. Moreover, to explore whether

```
SELECT
    L_ORDERKEY,
    SUM(L_EXTENDEDPRICE*(1-L_DISCOUNT)) AS REVENUE,
    O_ORDERDATE,
    O_SHIPPRIORITY
FROM
    CUSTOMER,
    ORDERS,
    LINEITEM
WHERE
    C_MKTSEGMENT = 'BUILDING'
    AND C_CUSTKEY = O_CUSTKEY
    AND L_ORDERKEY = O_ORDERKEY
    AND O_ORDERDATE < date'1995-03-15'
    AND L_SHIPDATE > date'1995-03-15'
    GROUP BY L_ORDERKEY,
    O_ORDERDATE,
    O_SHIPPRIORITY
    ORDER BY REVENUE DESC,
    O_ORDERDATE;
```

(a) Q3

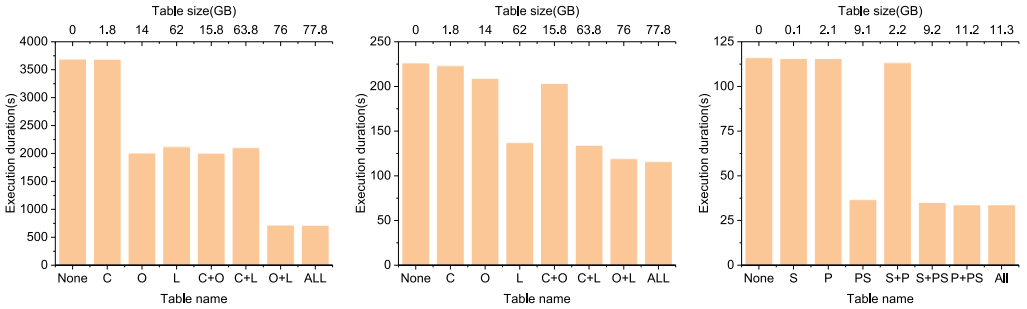
```
SELECT
    C_CUSTKEY, C_NAME,    SUM(L_EXTENDEDPRICE * (1 -
    L_DISCOUNT)) AS REVENUE, C_ACCTBAL, N_NAME,
    C_ADDRESS, C_PHONE, C_COMMENT
FROM
    CUSTOMER,
    ORDERS,
    LINEITEM,
    NATION
WHERE
    C_CUSTKEY = O_CUSTKEY
    AND L_ORDERKEY = O_ORDERKEY
    AND O_ORDERDATE >= '1993-10-01'
    AND O_ORDERDATE < '1993-10-01'+ interval '3' month
    AND L_RETURNFLAG = 'R'
    AND C_NATIONKEY = N_NATIONKEY
GROUP BY
    C_CUSTKEY, C_NAME, C_ACCTBAL, C_PHONE,
    N_NAME, C_ADDRESS, C_COMMENT
ORDER BY
    REVENUE DESC;
```

(b) Q10

```
SELECT
    P_BRAND, P_TYPE, P_SIZE,
    COUNT(DISTINCT PS_SUPPKEY) AS SUPPLIER_CNT
FROM
    PARTSUPP,
    PART
WHERE
    P_PARTKEY = PS_PARTKEY AND P_BRAND <> 'BRAND#45'
    AND P_TYPE NOT LIKE 'MEDIUM POLISHED%'
    AND P_SIZE IN (49, 14, 23, 45, 19, 3, 36, 9)
    AND PS_SUPPKEY NOT IN
(SELECT
    S_SUPPKEY
FROM
    SUPPLIER
WHERE
    S_COMMENT LIKE '%Customer%Complaints%'
)
GROUP BY
    P_BRAND, P_TYPE, P_SIZE
ORDER BY
    SUPPLIER_CNT DESC, P_BRAND, P_TYPE, P_SIZE;
```

(c) Q16

Fig. 14. Query clause of Q3, Q10, and Q16.



(a) Q3 (b) Q10 (c) Q16

Fig. 15. The execution duration for the hybrid table placement algorithm. Note, the abbreviation of each table's name and its corresponding size are listed in Table 3.

the query performance improvement is proportional to the Optane SSD's utilization size, the tables required by each query were placed on the Optane SSD with different combinations and their corresponding execution durations recorded. Figure 15 shows a plot of the experimental results, where the bottom x-axis represents the various table placement scenarios on the Optane SSD. For instance, as in Figure 15(a), "None" means that no table was stored in the Optane SSD during the query, "C" represents that only the CUSTOMER table was stored in the Optane SSD, "C+O" represents that the tables CUSTOMER and ORDERS were placed in the Optane SSD, and "ALL" means that all three required tables, including CUSTOMER, ORDERS, and LINEITEM, were placed on the Optane SSD. The top x-axis represents the space required to store the target tables on the Optane SSD, in gigabytes. The y-axis records the execution duration of the various table placement combinations. Analyzing the results revealed that when the Optane SSD's utilization space is certain, the performance improvement delivered by the hybrid table placement algorithm for each query type may be different from the conclusion drawn in the previous section. In Section 5, it was found that when the Optane SSD is used as a storage device to store all database tables, the Optane SSD delivers the highest performance accelerations for randomly and repeatedly read queries, a relatively lower speedup for randomly read queries, and the lowest improvement for sequentially read queries. However, for the hybrid table placement algorithm, this conclusion may not always be true. For instance, when the Optane SSD utilization size is set to 14GB, the hybrid system delivers the highest performance acceleration, 3.4X, for randomly read query Q16 (in this scenario, all tables required by Q16 are stored in the Optane SSD); a relatively lower speedup, 1.8X,

Table 6. Characteristics of Q3

	Sequential Read%	Random Read%	Accessed Data Size (GB)	Repeat Access Rate
CUSTOMER	99.6%	0.4%	1.8	1
ORDERS	8.5%	91.5%	145.6	10.4
LINEITEM	2.0%	98.0%	138.5	2.2

to randomly and repeatedly read query Q3; and the lowest improvement, 1.1X, to sequentially read query Q10. In addition, it was noted that the performance acceleration provided by the hybrid table placement algorithm was not always proportional to the Optane SSD's utilization size, especially for randomly and repeatedly read queries. For instance, as shown in Figure 15(a), the execution duration, 2,119.38*s*, of placing a larger table, LINEITEM (62GB), on Optane SSD for Q3 lags even behind the execution duration, 2,010.04*s*, of buffering a smaller table, ORDERS (14GB). In other words, for the hybrid table placement algorithm, using a larger Optane SSD space does not always provide performance gains.

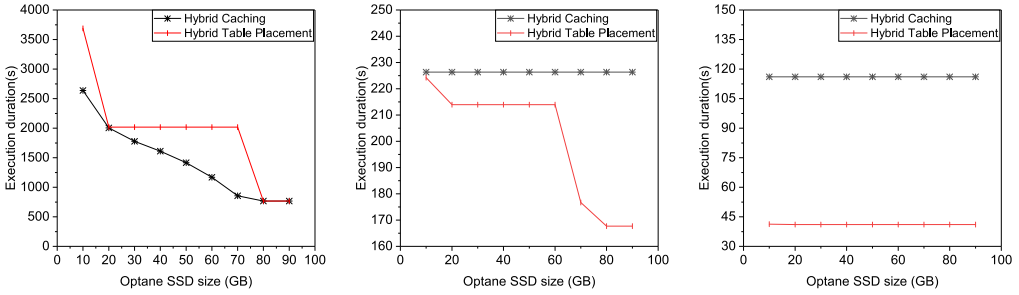
To find the reasons that underlie these results, an effort was made to explore more details for Q3. The sequential and random access percentages, the total accessed data size, and the repeat access rate were recorded for each table required by Q3 in Table 6. From the results, it was noted that Q3's performance was mainly limited by two randomly and repeatedly read tables, ORDERS and LINEITEM. During the query, 145GB and 138.5GB of data were randomly read from these two tables, respectively. Even if one table, ORDER, is placed on the Optane SSD, Q3's performance is still limited by randomly reading 138.5GB of data from the LINEITEM table on the NAND SSD. As a result, when the Optane SSD utilization space was set to 14GB, the hybrid system delivered only 1.8X speedup for Q3, which was much less than its performance acceleration, 3.4X, on a randomly read query, Q10. On the other hand, because Q3 read more data from the ORDERS table, 145GB, than from the LINEITEM table, 138.5GB, placing the smaller table, ORDERS, on the Optane SSD led to greater performance improvement than buffering the larger LINEITEM table.

By summarizing these experimental results, it can be concluded that (1) the hybrid table placement algorithm may not always deliver the greatest performance improvement for randomly and repeatedly read queries compared with the other two types of query; (2) for the hybrid table placement algorithm, it is not always true that placing a larger table on SSD or using more Optane SSD space can deliver higher performance to a database query; (3) hence, when designing a hybrid table placement algorithm for a database application, the impact of both the access patterns and the repeat access rates of the database tables must be carefully considered to deliver performance improvement.

6.2 Hybrid System with Caching Algorithm

We discussed in the previous subsection that for some database queries, such as Q3, the pages within the table would be repeatedly accessed during querying. Thus, admitting those pages that have temporal locality is expected to accelerate a query's performance once they are referenced again. In this subsection, we will study the following two questions: (1) How does the hybrid caching algorithm impact the performance of the three types of queries, including sequentially read queries, randomly read queries, and randomly and repeatedly read queries? (2) Compared with the hybrid table placement algorithm, does the hybrid caching algorithm always deliver better performance?

To address these questions, we use the same three representative queries as in the previous subsection. These queries include Q3, a randomly and repeatedly read query; Q10, a randomly read



(a) Q3 (b) Q10 (c) Q16

Fig. 16. The execution duration for the hybrid caching algorithm.

query; and Q16, a sequentially read query. The flashcache [38] has been built on our system, where the flashcache is a block cache component that uses fast storage device as a block cache for slow ones. The block cache is managed by an LRU-like caching algorithm. In the hybrid system, the application, MySQL, is running on the top. While I/O requests are sent into the block layer, flashcache remaps I/O into either block cache or storage devices based on its internal caching algorithm. In our experiment, we modified the original LRU-like caching algorithm to LRU and use Optane SSD as a block cache for NAND SSD. We record the execution duration of each representative query under different Optane SSD sizes, from 10GB to 90GB, with granularity of 10GB. For comparison, the hybrid table placement algorithms' results for three representative queries are also shown. The experimental results are presented in Figure 16. The x-axis represents Optane SSD size. The y-axis records the execution duration. Note here, for the hybrid table placement algorithm, we know that it is not always true that placing a larger table on SSD or using more Optane SSD space can deliver higher performance to a database query. In the experiment, while the Optane SSD size is fixed, we always record the best performance for different table combinations' results. For instance, for Q3, while the Optane SSD size is set to 70GB, the different table combinations that can be placed into Optane SSD are table C, 1.3GB; table O, 14GB; table L, 62GB; tables C+O, 15.8GB; and tables C+L, 63.8GB. Placing table C+O delivers the best performance, 2,006.7s, compared with others. When the Optane SSD size is set to 70GB, the execution duration of the hybrid table placement algorithm for Q3 is 2,006.7s. Moreover, the table migration time for migrating target table(s) into Optane SSD for the hybrid table placement algorithm has also been included in its overall performance.

As Figure 16 illustrated, for the randomly and repeatedly read query, Q3, while the Optane SSD size is between 10GB and 80GB, the hybrid caching algorithm delivers better performance than the hybrid table placement algorithm. This performance gap is continuously increasing as we increase the Optane SSD size from 20GB to 70GB. This is because a table is the smallest caching unit for the hybrid table placement algorithm. While the available Optane SSD space is less a target table's size, since the target table cannot be temporally cached, the hybrid table placement algorithm is not able to deliver any performance gains. This performance issue is reflected by a flat line in Figure 16(a) when the Optane SSD size is set to 20GB to 70GB. In contrast, for the hybrid caching algorithm, since a page is the smallest caching unit, the larger Optane SSD size allows more repeatedly accessed pages to be cached, and, as a result, more performance gains are delivered for Q3. As we continuously increase the Optane SSD size to 80GB, since all required tables of Q3 can be temporally cached, the hybrid table placement algorithm provides similar performance to the hybrid caching algorithm.

For Q10, a randomly read query, and Q16, a sequentially read query, since all pages are only accessed once during querying, no performance gains are provided by the hybrid caching

algorithm. To the contrary, we observed significant performance improvement of the hybrid table placement algorithm for Q10 and Q16. There are two reasons to explain the above results. First, Optane SSD delivers much higher performance than NAND SSD when processing multiple I/O threads, as discussed in Section 4.3. Second, because of the read-ahead mechanism, NAND SSD provides similar performance as Optane SSD for single-thread sequential read operation, as discussed in Section 4.1. Thus, the extra table migration time consumed by the hybrid table placement algorithm is relatively small compared to its performance gains. For the above two reasons, the hybrid table placement algorithm accelerates both randomly read and sequentially read queries' performance significantly. After we increased the Optane SSD size to 20GB, since all required tables of Q16 have to be temporarily buffered, the hybrid table placement algorithm's performance was not changed anymore as we continued to increase the Optane SSD size.

Summarizing our experimental results, it can be concluded that (1) the hybrid caching algorithm does not always outperform the hybrid table placement algorithm, and the performance gains provided by the two types of hybrid algorithms are highly dependent on the query's type, and (2) hence, when designing a hybrid system for a specific database query workload, the queries' characteristics must be carefully considered to decide which hybrid algorithm should be employed.

7 RELATED WORK

The NAND SSD has been an active research area in recent decades. Many research studies have been conducted on SSD performance evaluation and optimization. For example, Agrawal et al. [8] provided an overview of SSD internal origination and discussed the design tradeoffs achievable by a trace-driven simulator. They found that SSD performance was highly sensitive both to internal hardware design and to software implementation. Dirik and Jacob [13] studied the internal parallelism inside SSDs and concluded that exploiting SSD internal concurrency would deliver significant performance improvement. Chen et al. [11, 12] explored the unique performance behavior and internal characteristics of a SATA-based NAND SSD using a number of custom-defined microbenchmarks. Similarly, Kim et al. [29] presented a number of methodologies for extracting the key configurations inside a SATA-based NAND SSD, including cluster page size, cluster block size, and so on. By optimizing the operating system through these extracted parameters, the overall system performance could be greatly accelerated.

To improve performance, recent research has focused mainly on addressing the lack of NAND SSD data update-in-place ability through either a flash file system implementation [3, 16, 48] or a flash translation layer design [27, 35]. Josephson et al. [23] introduced a direct file system (DFS). Using a newly designed virtual flash storage layer, the DFS can access flash memory cells directly. As a result, DFS is substantially faster than other conventional file systems. Lu et al. [36] proposed an object-based flash translation layer (OFTL). With their design, storage management is assigned to the NAND SSD FTL from the file system to manage flash memory directly. Benefiting from this advanced implementation, write amplifications were reduced tremendously. Taking a different approach, Soundararajan et al. [44] developed a hybrid storage system, Griffin, which used an HDD as a write cache for a NAND SSD. During I/O processing, the HDD maintained a log-structured cache and wrote cached data periodically to the SSD. With this implementation, Griffin not only reduced the number of writes into flash memory but also provided similar performance to a NAND SSD.

By combining high-performance 3D Xpoint nonvolatile memory media with a well-optimized solid-state drive, the Optane SSD can outperform a conventional NAND SSD in multiple aspects. There has been significant prior work studying the Optane SSD. Hady et al. [18] and Izraelevitz et al. [21] provided basic performance evaluations of Optane SSD. The performance impacts of Optane SSD to real applications have been studied as well. By using multiple benchmarks, Zhang et al. [51] analyzed Optane SSD performance behaviors on virtualized and nonvirtualized

environments, respectively. A hybrid system called MyNVM for using Optane SSD as a second-level cache for key-value store has been proposed in [14]. MyNVM reduces DRAM footprint as well as keeping up competitive performance. However, previous work has provided limited information about Optane SSD's intrinsic characteristics. In this study, through carefully designed microbenchmarks, multiple Optane SSD characteristics have been explored, while its potential benefits were confirmed using a real database application.

8 CONCLUSIONS

Through intensive experiments and measurements, this study found the following: (1) By using a similar internal architecture, Optane SSD exhibits the same performance trend as NAND SSD: either larger request size or larger queue depth can increase Optane SSD's performance due its internal parallelism. On the other hand, benefits from employing the high-performance 3D Xpoint memory media, Optane SSD delivers substantial performance improvement compared with conventional NAND flash-based SSD. This performance advantage allows Optane SSD to provide a bridge between slower storage devices and high-performance DRAM. (2) Even though 3D Xpoint is a byte-addressable memory, because it is integrated into a block-level storage device, any request submitted to Optane SSD must be aligned with a page size to avoid performance degradation. (3) Because 3D Xpoint memory is byte addressable and supports data update-in-place, some of the common problems that typically exist within the conventional NAND SSD, including read-modify-write and write-driven garbage collection, have been removed from Optane SSD. Thus, designing a hybrid storage system using Optane SSD as a write buffer for NAND SSD to improve NAND SSD's performance and lifespan could become a potential research direction in the future. (4) The read disturb issue is one of the key factors that contributes to the long tail latency in the Optane SSD. For modern applications that have strict requirements on I/O latency distribution, how to mitigate Optane SSD's performance variation due to its read disturb issue is a potentially significant research direction. (5) When designing the hybrid Optane and NAND SSD storage system for a database application, the characteristics of database query workload are key factors that must be considered carefully to decide which hybrid algorithm should be used to deliver better performance: either the hybrid table placement algorithm or hybrid caching algorithm. In general, the present research provides new insight into the Optane SSD and studies its implications for a real database system. It is hoped that these experimental results can inspire researchers in OS design and system optimization and help them rethink their storage hierarchy designs in their further research.

REFERENCES

[1] Transaction Processing Performance Council. *TPC-H Benchmark*. http://www.tpc.org/tpch/.
[2] Jens Axboe. 2016. Flexible I/O Tester. https://github.com/axboe/fio.
[3] Aleph One Company. 2012. *Yet Another Flash File System (YAFFS)*. https://yaffs.net/.
[4] Intel Corporation. 2014. Intel SSD DC P3700 Series. https://ark.intel.com/content/www/us/en/ark/products/series/79628/intel-ssd-dc-p3700-series.html.
[5] Oracle Corporation. 2017. *MySQL 5.7*. https://dev.mysql.com/doc/refman/5.7/en/.
[6] Intel Corporation. 2018. Product Brief: Intel Optane SSD DC P4800X Series. Retrieved from https://www.intel.com/content/www/us/en/solid-state-drives/optane-ssd-dc-p4800x-brief.html.
[7] Intel Corporation. 2018. World's Most Responsive Data Center SSD. Retrieved from https://www.intel.com/content/dam/www/public/us/en/documents/product-briefs/optane-ssd-dc-p4800x-brief.pdf.
[8] Nitin Agrawal, Vijayan Prabhakaran, Ted Wobber, John D. Davis, Mark S. Manasse, and Rina Panigrahy. 2008. Design tradeoffs for SSD performance. In *USENIX Annual Technical Conference*, Vol. 57.
[9] Alan D. Brunelle. 2006. Block I/O layer tracing: blktrace. *HP, Gelato-Cupertino, CA*.
[10] Mustafa Canim, George A. Mihaila, Bishwaranjan Bhattacharjee, Kenneth A. Ross, and Christian A. Lang. 2010. SSD bufferpool extensions for database systems. *Proceedings of the VLDB Endowment* 3, 1–2 (2010), 1435–1446.

[11] Feng Chen, David A. Koufaty, and Xiaodong Zhang. 2009. Understanding intrinsic characteristics and system implications of flash memory based solid state drives. In *ACM SIGMETRICS Performance Evaluation Review*, Vol. 37. ACM, 181–192.

[12] Feng Chen, Rubao Lee, and Xiaodong Zhang. 2011. Essential roles of exploiting internal parallelism of flash memory based solid state drives in high-speed data processing. In *IEEE 17th International Symposium on High Performance Computer Architecture (HPCA'11)*. IEEE, 266–277.

[13] Cagdas Dirik and Bruce Jacob. 2009. The performance of PC solid-state disks (SSDs) as a function of bandwidth, concurrency, device architecture, and system organization. In *ACM SIGARCH Computer Architecture News*, Vol. 37. ACM, 279–289.

[14] Assaf Eisenman, Darryl Gardner, Islam AbdelRahman, Jens Axboe, Siying Dong, Kim Hazelwood, Chris Petersen, Asaf Cidon, and Sachin Katti. 2018. Reducing DRAM footprint with NVM in Facebook. In *Proceedings of the 13th EuroSys Conference*. ACM, 42.

[15] Holloway H. Frost, Charles J. Camp, Timothy J. Fisher, James A. Fuxa, and Lance W. Shelton. 2010. Efficient reduction of read disturb errors in NAND flash memory. US Patent 7,818,525.

[16] Eran Gal and Sivan Toledo. 2005. A transactional flash file system for microcontrollers. In *USENIX Annual Technical Conference, General Track*. 89–104.

[17] Aayush Gupta, Youngjae Kim, and Bhuvan Urgaonkar. 2009. *DFTL: A Flash Translation Layer Employing Demand-Based Selective Caching of Page-level Address Mappings*, Vol. 44. ACM.

[18] Frank T. Hady, Annie Foong, Bryan Veal, and Dan Williams. 2017. Platform storage performance with 3D XPoint technology. *Proceedings of the IEEE* 105, 9 (2017), 1822–1833.

[19] Mingzhe Hao, Gokul Soundararajan, Deepak R. Kenchammana-Hosekote, Andrew A. Chien, and Haryadi S. Gunawi. 2016. The tail at store: A revelation from millions of hours of disk and SSD deployments. In *14th USENIX Conference on File and Storage Technologies (FAST'16)*. 263–276.

[20] Yang Hu, Hong Jiang, Dan Feng, Lei Tian, Hao Luo, and Chao Ren. 2013. Exploring and exploiting the multilevel parallelism inside SSDs for improved performance and endurance. *IEEE Transactions on Computers* 62, 6 (2013), 1141–1155.

[21] Joseph Izraelevitz, Jian Yang, Lu Zhang, Juno Kim, Xiao Liu, Amirsaman 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. *arXiv preprint arXiv:1903.05714* (2019).

[22] Seongwook Jin, Jaehong Kim, Jaegeuk Kim, Jaehyuk Huh, and Seungryoul Maeng. 2011. Sector log: Fine-grained storage management for solid state drives. In *Proceedings of the 2011 ACM Symposium on Applied Computing*. ACM, 360–367.

[23] William K. Josephson, Lars A. Bongo, Kai Li, and David Flynn. 2010. DFS: A file system for virtualized flash storage. *ACM Transactions on Storage (TOS)* 6, 3 (2010), 14.

[24] Dawoon Jung, Jeong-Uk Kang, Heeseung Jo, Jin-Soo Kim, and Joonwon Lee. 2010. Superblock FTL: A superblock-based flash translation layer with a hybrid address translation scheme. *ACM Transactions on Embedded Computing Systems (TECS)* 9, 4 (2010), 40.

[25] Jeong-Uk Kang, Heeseung Jo, Jin-Soo Kim, and Joonwon Lee. 2006. A superblock-based flash translation layer for NAND flash memory. In *Proceedings of the 6th ACM & IEEE International Conference on Embedded Software*. ACM, 161–170.

[26] Jesung Kim, Jong Min Kim, Sam H. Noh, Sang Lyul Min, and Yookun Cho. 2002. A space-efficient flash translation layer for CompactFlash systems. *IEEE Transactions on Consumer Electronics* 48, 2 (2002), 366–375.

[27] Jesung Kim, Jong Min Kim, Sam H. Noh, Sang Lyul Min, and Yookun Cho. 2002. A space-efficient flash translation layer for CompactFlash systems. *IEEE Transactions on Consumer Electronics* 48, 2 (2002), 366–375.

[28] Jaeho Kim, Yongseok Oh, Eunsam Kim, Jongmoo Choi, Donghee Lee, and Sam H. Noh. 2009. Disk schedulers for solid state drivers. In *Proceedings of the 7th ACM International Conference on Embedded Software*. ACM, 295–304.

[29] Jaehong Kim, Sangwon Seo, Dawoon Jung, Jin-Soo Kim, and Jaehyuk Huh. 2012. Parameter-aware I/O management for solid state disks (SSDs). *IEEE Transactions on Computers* 61, 5 (2012), 636–649.

[30] Michael Kund, Gerhard Beitel, C.-U. Pinnow, Thomas Rohr, Jorg Schumann, Ralf Symanczyk, K. Ufert, and Gerhard Muller. 2005. Conductive bridging RAM (CBRAM): An emerging non-volatile memory technology scalable to sub 20nm. In *IEEE International Electron Devices Meeting, 2005. IEDM Technical Digest*. IEEE, 754–757.

[31] Simone Lavizzari, Daniele Ielmini, Deepak Sharma, and Andrea Leonardo Lacaita. 2008. Transient effects of delay, switching and recovery in phase change memory (PCM) devices. In *IEEE International Electron Devices Meeting, 2008 (IEDM'08)*. IEEE, 1–4.

[32] Sang-Won Lee, Bongki Moon, and Chanik Park. 2009. Advances in flash memory SSD technology for enterprise database applications. In *Proceedings of the 2009 ACM SIGMOD International Conference on Management of Data*. ACM, 863–870.

[33] Sang-Won Lee, Bongki Moon, Chanik Park, Jae-Myung Kim, and Sang-Woo Kim. 2008. A case for flash memory SSD in enterprise database applications. In *Proceedings of the 2008 ACM SIGMOD International Conference on Management of Data*. ACM, 1075–1086.

[34] Sang-Won Lee, Dong-Joo Park, Tae-Sun Chung, Dong-Ho Lee, Sangwon Park, and Ha-Joo Song. 2007. A log buffer-based flash translation layer using fully-associative sector translation. *ACM Transactions on Embedded Computing Systems (TECS)* 6, 3 (2007), 18.

[35] Sang-Won Lee, Dong-Joo Park, Tae-Sun Chung, Dong-Ho Lee, Sangwon Park, and Ha-Joo Song. 2007. A log buffer-based flash translation layer using fully-associative sector translation. *ACM Transactions on Embedded Computing Systems (TECS)* 6, 3 (2007), 18.

[36] Youyou Lu, Jiwu Shu, and Weimin Zheng. 2013. Extending the lifetime of flash-based storage through reducing write amplification from file systems. In *11th USENIX Conference on File and Storage Technologies (FAST'13)*, Vol. 13.

[37] Tian Luo, Rubao Lee, Michael Mesnier, Feng Chen, and Xiaodong Zhang. 2012. hStorage-DB: Heterogeneity-aware data management to exploit the full capability of hybrid storage systems. *Proceedings of the Conference on Very Large Data Bases (VLDB) Endowment* 5, 10 (2012), 1076–1087.

[38] Paul Saab Mohan Srinivasan. 2010. Flashcache. Retrieved from https://github.com/facebookarchive/flashcache.

[39] Mark Moshayedi and Patrick Wilkison. 2008. Enterprise SSDs. *Queue* 6, 4 (2008), 32–39.

[40] Prashant J. Nair, Chiachen Chou, Bipin Rajendran, and Moinuddin K. Qureshi. 2015. Reducing read latency of phase change memory via early read and turbo read. In *2015 IEEE 21st International Symposium on High Performance Computer Architecture (HPCA'15)*. IEEE, 309–319.

[41] Hiroyuki Nakamoto, Daisuke Yamazaki, Takuji Yamamoto, Hajime Kurata, Satoshi Yamada, Kenji Mukaida, Tsuzumi Ninomiya, Takashi Ohkawa, Shoichi Masui, and Kunihiko Gotoh. 2007. A passive UHF RF identification CMOS tag IC using ferroelectric RAM in 0.35-um technology. *IEEE Journal of Solid-State Circuits* 42, 1 (2007), 101–110.

[42] Dushyanth Narayanan, Austin Donnelly, and Antony Rowstron. 2008. Write off-loading: Practical power management for enterprise storage. *ACM Transactions on Storage (TOS)* 4, 3 (2008), 10.

[43] Seon-yeong Park, Euiseong Seo, Ji-Yong Shin, Seungryoul Maeng, and Joonwon Lee. 2010. Exploiting internal parallelism of flash-based SSDs. *IEEE Computer Architecture Letters* 9, 1 (2010), 9–12.

[44] Gokul Soundararajan, Vijayan Prabhakaran, Mahesh Balakrishnan, and Ted Wobber. 2010. Extending SSD lifetimes with disk-based write caches. In *USENIX Conference on File and Storage Technologies (FAST'10)*, Vol. 10. 101–114.

[45] Hermann Strass. 2016. *An Introduction to NVMe*. https://labs.seagate.com/wp-content/uploads/sites/7/2018/09/an-introduction-to-nvme-tp690-1-1605us.pdf.

[46] Beth Trushkowsky, Peter Bodík, Armando Fox, Michael J. Franklin, Michael I. Jordan, and David A. Patterson. 2011. The SCADS director: Scaling a distributed storage system under stringent performance requirements. In *USENIX Conference on File and Storage Technologies (FAST'11)*, Vol. 11. 163–176.

[47] H.-S. Philip Wong, Simone Raoux, SangBum Kim, Jiale Liang, John P. Reifenberg, Bipin Rajendran, Mehdi Asheghi, and Kenneth E. Goodson. 2010. Phase change memory. *Proceedings of the IEEE* 98, 12 (2010), 2201–2227.

[48] David Woodhouse. 2001. JFFS: The journalling flash file system. In *Ottawa Linux Symposium*, Vol. 2001.

[49] 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*. ACM, 6.

[50] Jinfeng Yang and David J. Lilja. 2018. Reducing relational database performance bottlenecks using 3D XPoint storage technology. In *2018 17th IEEE International Conference on Trust, Security and Privacy in Computing and Communications/12th IEEE International Conference on Big Data Science and Engineering (TrustCom/BigDataSE'18)*. IEEE, 1804–1808.

[51] Jiachen Zhang, Peng Li, Bo Liu, Trent G. Marbach, Xiaoguang Liu, and Gang Wang. 2018. Performance analysis of 3D XPoint SSDs in virtualized and non-virtualized environments. In *2018 IEEE 24th International Conference on Parallel and Distributed Systems (ICPADS'18)*. IEEE, 1–10.

Received April 2019; revised September 2019; accepted November 2019