Characterizing and Optimizing Remote Persistent Memory with RDMA and NVM

Xingda Wei, Xiating Xie, Rong Chen, Haibo Chen, and Binyu Zang,
Shanghai Jiao Tong University; Shanghai AI Laboratory;
Engineering Research Center for Domain-specific Operating Systems

https://www.usenix.org/conference/atc21/presentation/wei

This paper is included in the Proceedings of the
2021 USENIX Annual Technical Conference.

July 14–16, 2021

978-1-939133-23-6

Open access to the Proceedings of the
2021 USENIX Annual Technical Conference
is sponsored by USENIX.

Characterizing and Optimizing Remote Persistent Memory with RDMA and NVM

Xingda Wei, Xiating Xie, Rong Chen, Haibo Chen, Binyu Zang

Institute of Parallel and Distributed Systems, Shanghai Jiao Tong University
Shanghai Artificial Intelligence Laboratory
Engineering Research Center for Domain-specific Operating Systems, Ministry of Education, China

Abstract

The appealing properties of NVM including high performance, persistence, and byte-addressability, and a recent active thread of building remote memory systems with RDMA, have produced considerable interest in combining them for fast and persistent remote memory systems. However, many prior systems are either based on emulated NVM or have failed to fully exploit NVM characteristics, leading to suboptimal performance.

This paper conducts a systematic study to summarize optimization hints that the system designer can use to exploit NVM with RDMA better. Specifically, we demonstrate how system configurations, NVM access patterns, and RDMA-aware optimizations affect the efficacy of RDMA-NVM systems. Based on the summarized hints, we empirically study the design of two existing RDMA-NVM systems, namely a distributed database (DrTM+H) and a distributed file system (Octopus). Both systems are designed when no production NVM is available, and neither of them achieves good performance on it. Our optimized systems achieve up to 2.4X (from 1.2X) better performance.

1 Introduction

People have been studying systems with emerging hardware technologies like Remote Direct Memory Access (RDMA) and Non-Volatile Memory (NVM) for many years, including but not limited to databases [8, 11, 25, 37, 39, 53], file systems [31, 32, 58], key-value stores [2, 10], and distributed shared memory systems [33, 38, 61]. These RDMA-NVM systems can either leverage NVM to persistently store the data or use it as DRAM to extend DRAM capacity.

Unfortunately, few systematic studies examined how to best leverage NVM with RDMA, since production NVM is only publicly available via Intel Optane DC persistent memory [36] (Optane PM[1]) until recently. Except for a few systems [22, 33], prior work either uses emulated NVM [58, 61] or simply treats DRAM as NVM [8, 11, 32, 53]. Without such a study, system developers are unclear whether existing RDMA-NVM designs are efficient for Optane PM due to the following three reasons. First, emulating NVM with RDMA is particularly challenging because, to the best of our knowledge, most NVM emulators use CPU for the emulation [49]. However, RDMA may access NVM in a CPU-bypassing manner. Second, a recent study revealed that even the emulator could not faithfully simulate many Optane PM features [59]. Finally, a few systems evaluated with Optane PM do not consider its unique performance characteristics [33].

Our initial experiments further demonstrate that RDMA-NVM systems suffer from inferior performance when they treat NVM as DRAM. For example, the performance of *remote write* is far from the limits of NVM (§3) after switching the memory used in a *remote write* benchmark from DRAM to NVM: 16B one-sided RDMA WRITE only achieves 29% of NVM's ideal write throughput. Hence, it is imperative to conduct a thorough study of the inferior performance and provide optimizations to mitigate the inefficiency.

There have been valuable studies on how to efficiently use NVM [59] with CPU and how CPU cache may affect the efficiency of RDMA with NVM [22]. Yang *et al*. [59] provide optimizations for the CPU to best utilize Optane PM. We study their findings on RDMA-NVM systems and confirm the importance of their optimizations. Nevertheless, we found some of the optimizations are suboptimal when considering RDMA. We further present optimizations that are best suited for RDMA on NVM. Kalia *et al*. [22] is the most relevant work: we share the same goal of improving RDMA's performance with NVM. Specifically, they identify how CPU cache could hinder RDMA from fully utilizing NVM write bandwidth. We made a similar observation during our study. Besides, we also study other RDMA-NVM related factors, including inappropriate system configurations (§4.1) and application access patterns (§4.2).

Finally, existing systems use two network roundtrips to implement persistent write atop RDMA and NVM [20] because existing RDMA hardware is unaware of NVM. We ar-

[1]Since we exclusively study Optane PM in this paper, we use the terms NVM and Optane PM interchangeably throughout the paper.

gue that RDMA-NVM systems should consider broadly explored RDMA-aware optimizations [23, 24] to improve the persistent write performance with RDMA. With the help of known RDMA-aware optimizations, RDMA only needs one roundtrip for remote persistent write on the current hardware platforms (§4.3).

In this paper, we conduct a thorough systematic study on how to best utilize NVM with RDMA. Note that our focus is on *remote write*, i.e., the client issues write to the server NVM, either using one-sided or two-sided RDMA. The remote NVM read performance is close to that of DRAM (§3). To summarize, the contributions of this paper are:

A summary of optimization hints (H1–H9) to best utilize NVM with RDMA (§4). We study and collect various RDMA-NVM related optimizations—scattered from different sources—into one systematic study. We also propose new optimizations (**H6–H8**) that address the limitations of the existing study. The summarized hints are categorized into system configuration advice (§4.1), access pattern advice (§4.2) and RDMA-aware advice (§4.3). We empirically demonstrate how these hints help to fully utilize NVM for different RDMA primitives, i.e., RDMA can attain close to NVM write bandwidth and processing power.

An end-to-end study of improved RDMA-NVM system designs (§6). We use the summarized hints to analyze and improve the design of existing RDMA-NVM systems, namely a distributed database (DrTM+H [53]) and a distributed file system (Octopus [32]). We find that there is still significant room for improvement because both of them are designed when no production NVM is available. Our study helps to improve the throughput of DrTM+H by 1.5X and 2.2X for TPC-C [46] new-order transaction and SmallBank [45], respectively. It also improves the I/O throughput of file data operations in Octopus by up to 2.4X (from 1.2X). These results strongly suggest we need further revisiting the design and implementations of existing RDMA-NVM systems, especially those not designed for Optane PM.

Our tools and benchmarks are available at `https://github.com/SJTU-IPADS/librdpma`.

2 Background

Figure 1 presents typical hardware components of a node with NVM in an RDMA-capable cluster. RDMA-capable NIC (RNIC) and NVM are attached to the processor, and they communicate with each other via the PCI Express (PCIe) bus.

2.1 Optane PM (NVM)

Intel Optane DC persistent memory [36] (Optane PM) is the first commercially available NVM. Besides a huge

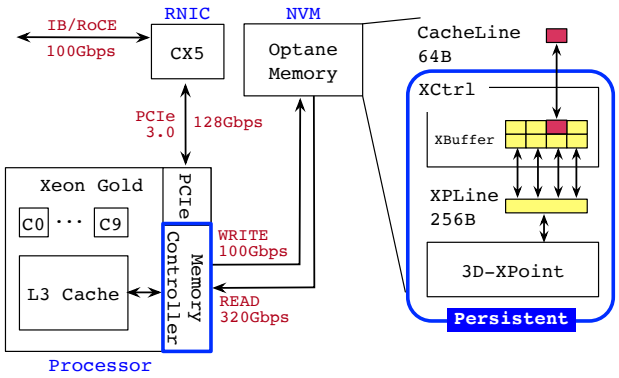

Figure 1. *Hardware components of a node with NVM in an RDMA-capable cluster.*

performance gain compared to traditional persistent storage (e.g., SSD), Optane PM also provides a DRAM-like memory interface. Thus, CPU can use `load` and `store`/`non-temporal store`[2] (`ntstore`) to read and write it, and RNIC can access it through PCIe read/write transactions.

Our study relies on an in-depth look at how Optane PM handles reads/writes. The right half of Figure 1 presents an overview of its components. Data is stored in NVM DIMMs (3D XPoint), while XController (XCtrl) transforms the read-/write requests from the processor/PCIe into read/write requests to 3D XPoint. XCtrl has two important features. First, it receives requests in cacheline (CLine) granularity (64B), while 3D XPoint stores data in XPLine granularity (256B). Such a difference in granularity may incur read/write amplification. Second, in order to reduce write amplification, XCtrl has a small write-combining buffer (XBuffer) that merges adjacent cacheline writes into one XPLine write. Note that read requests also compete for XBuffer with write requests [59].

Persistent domain. Data is persistent once it reaches the node's persistent domain. On the current hardware platform, the persistent domain comprises the Optane PM and the processor's memory controller, as shown in Figure 1. Future hardware will further extend the persistent domain to the processor cache [4]. Nevertheless, the scope of the persistent domain is orthogonal to the results of this paper. We describe its impact in §5 in more detail.

Asymmetric performance feature. Optane PM is known for two asymmetric performance features. First, its read is much faster than write (320Gbps vs. 100Gbps). Second, sequential write has a higher bandwidth than random write due to the involvement of XBuffer.

Optane PM counters. Optane PM provides various useful counters[3] that we use to analyze its behavior: `NReadReq` and

[2] `non-temporal store` has the same semantic as `store` except that it bypasses the CPU cache.
[3] Measured via `ipmctl` [5].

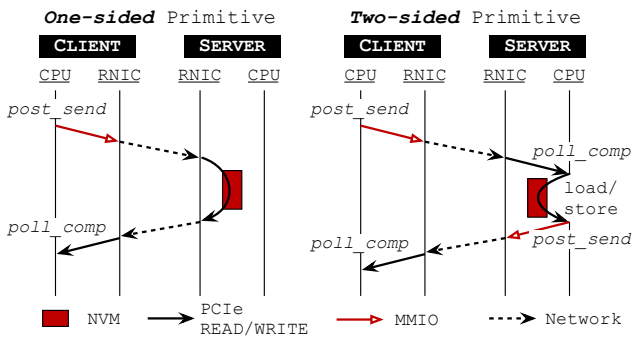

Figure 2. *An overview of the interaction between RDMA and NVM.*

`NWriteReq` record how many 64B read and write requests are received by XCtrl, while `NMediaRead` and `NMediaWrite` record how many bytes are read/written by 3D-XPoint Media. Based on these counters, we calculate the counter rates and use the counter rates to compute the read/write amplification of NVM: e.g., `NMediaWrite` *rate* / (`NWriteReq` *rate* $\times$64) measures the write amplification of Optane PM.

2.2 Remote direct memory access (RDMA)

RDMA is a fast networking feature with high throughput (e.g., 100Gbps bandwidth), low latency (e.g., 2μs), and low CPU overhead. Representative implementations of RDMA include InfiniBand (IB) and RDMA over Converged Ethernet (RoCE). RDMA is well-known for its one-sided primitives (READ/WRITE[4]): RDMA-capable NIC (RNIC) can directly read/write the server memory bypassing the server CPU. RDMA also provides two-sided primitives (SEND/RECV) that are similar to message passing.

QP and the programming model of RDMA. RDMA hosts use queue pair (QP) to issue RDMA requests. The QP contains one *send queue* and one *completion queue*. To issue an RDMA request (e.g., one-sided RDMA READ), the host calls *post_send*, which uses memory-mapped IO (MMIO) to post the request to the *send queue*. If the host marks the request as *signaled*, then it can further obtain the completion event of the sent request, e.g., whether the payload of the READ has been fetched to the host, by polling the *completion queue* via *poll_comp*.

2.3 RDMA with NVM

Figure 2 shows how various RDMA primitives interact with Optane PM. One-sided RDMA primitives communicate with Optane PM through PCIe read/write transactions, while two-sided RDMA uses server CPU to read/write Optane PM[5].

[4] We may use READ/WRITE as one-sided RDMA READ/WRITE.

[5] Although two-sided RDMA can use PCIe read/write transactions to write messages to Optane PM, we omit the discussion of such a case because its mechanism is the same as one-sided RDMA WRITE.

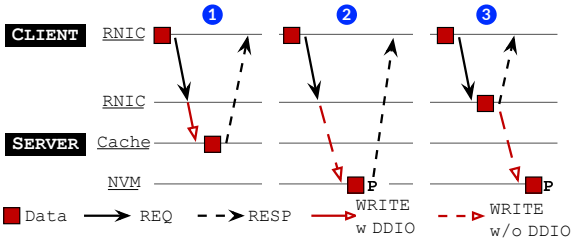

Figure 3. *Three execution flows of using one-sided RDMA to write the server NVM. Note that without proper configurations (see §2.3), all three request flows are possible. The client uses the MMIO to post the WRITE request (**REQ**), and the client RNIC generates the response (**RESP**) via DMA. When DDIO (§2.3) is enabled, RNIC writes to the server's last level cache (LLC). Otherwise, RNIC directly writes to Optane PM. **P** denotes the persistent point of the data, i.e., when the client can ensure the WRITE is persistent.*

When different RDMA primitives access NVM, several factors may impact their efficacy:

Access granularity. RNIC, CPU and NVM (including XController and 3D XPoint) have different access granularities. Requests that do not match the device granularity cause extra read/write requests to the NVM. Hence, systems should carefully tune their NVM access patterns for different RDMA primitives (§4.2). Table 1 summarizes the access granularities of different devices.

Table 1: *Access granularities of different hardware components.*

	CPU	PCIe	XController	3D XPoint
Granularity	CLine	CLine	CLine	XPLine
Payload	64B	64B	64B	256B

DDIO. Data Direct I/O (DDIO) [16] aims to improve the server cache locality of the DMA-ed data, which allows the last level cache (i.e., L3 Cache) as the primary destination of the RNIC's DMA-ed data (e.g., one-sided RDMA WRITE and two-sided RDMA). However, it is not friendly to Optane PM (§4.1).

Persistence. Two-sided primitives can use extended CPU instructions (e.g., `clwb`) to ensure that the write to NVM is persistent. However, one-sided RDMA has no such instruction. Thus, one-sided RDMA-NVM WRITE is not persistent.

Figure 3 depicts three possible execution flows of one-sided RDMA-NVM WRITE on the current hardware platforms, only in the second case that the data is persistent (❷). In the first case (❶), the data is not persistent because it still resides in the volatile processor cache when the client receives the response. In the third case (❸), the data is cached at the RNIC's internal buffer when the client believes the write has finished. RNIC is not in the persistent domain (see Figure 1).

Table 2: *The configurations of machines in our testbed.*

Name	#	Hardware
Server	1	2x Intel Xeon Gold 5215M (10 cores), 384GB DRAM
		2x ConnectX-5 IB RNIC (100Gbps)
		1x 1.5T NVM (12x Optane DIMM)
Client	5	2x Intel Xeon E5-2650 v4 (12 cores), 128GB DRAM
		2x ConnectX-4 IB RNIC (100Gbps)

Implementing persistent one-sided RDMA WRITE over current hardware (i.e., guarantee to achieve ❷ in Figure 3) requires specific configurations and extra one-sided requests: we should first disable DDIO to bypass the processor cache and then send an extra one-sided RDMA READ to the same QP issued the WRITE [19] to flush the previously cached WRITEs. These two steps guarantee the WRITE is executed as the second case in Figure 3. However, a strawman implementation of this strategy uses two network roundtrips for a single write [20]. We describe optimizations for persistent WRITE in §4.3.

3 Testbed and Methodology

Testbed. Table 2 lists the hardware descriptions of our testbed. The server machine that equips Optane PM has two 10-core Intel Xeon Gold 5215M processors, 384GB DRAM and two ConnectX-5 MT27800 100Gbps Infiniband NIC. We attach six NVM DIMMs to each server's processor, allowing them to achieve the maximum (ideal) bandwidth of Optane PM (320Gbps for read and 100Gbps for write). Each client machine has two 12-core Intel Xeon E5-2650 v4 processors, 128GB of DRAM, and two ConnectX-4 MCX455A 100Gbps Infiniband NIC. All machines are connected to a Mellanox SB7890 100Gbps InfiniBand Switch.

Target systems and evaluation methodology. The focus of this paper is on *remote write*, i.e., the client issues write requests to the server NVM using either one-sided or two-sided RDMA. For *remote read*, we find it has close performance to that of DRAM due to the asymmetric read/write performance feature (§2.1) of NVM.

To empirically analyze the read/write features of RDMA with NVM, we conduct a microbenchmark to evaluate the performance of *remote read* and *remote write* implemented by different RDMA primitives. In this benchmark, each client sends read/write requests with different payloads to the server's NVM via RDMA, similar to prior work [10, 23, 40, 53]. The request addresses are chosen randomly. For one-sided RDMA, the client directly uses its primitives to implement *remote read* and *remote write*. For two-sided RDMA, the client sends messages to the server, and the server reads/writes NVM with `memcpy` after receiving the messages. We implement the benchmark on a state-of-the-art distributed execution framework designed for RDMA [53]. Unless otherwise mentioned, we report the per-socket peak throughput or bandwidth of reading/writing NVM through RDMA.

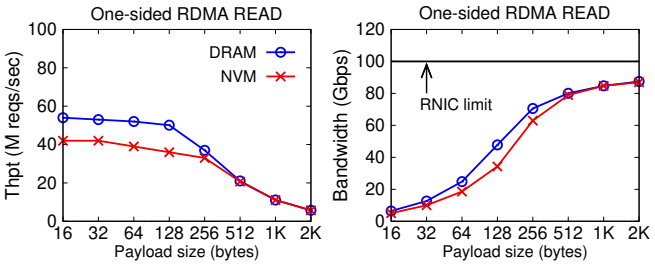

Figure 4. *A comparison of one-sided RDMA READ performance on DRAM and NVM, (a) throughput and (b) aggregated bandwidth.*

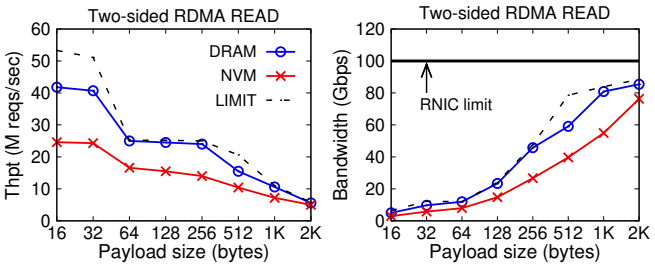

Figure 5. *A comparison of two-sided RDMA READ performance on DRAM and NVM, (a) throughput and (b) aggregated bandwidth. LIMIT is measured when the server directly returns to the client without reading the payload.*

Figure 4 and Figure 5 present the performance of *remote read* on DRAM and NVM for one-sided and two-sided RDMA, respectively. For large payloads (e.g., 2,048B), the read performance of NVM is close to that of DRAM for both one-sided and two-sided primitives (99% and 90%). Reading NVM can hardly become a bottleneck in RDMA-NVM systems since NVM has a much higher read bandwidth than RNIC (320Gbps vs. 100Gbps). Note that for small reads (e.g., 16B), two-sided RDMA READ still suffers from obvious throughput degradation: it only achieves 59% of the DRAM read throughput. This is because CPU has much a higher read latency when reading NVM compared to DRAM (271ns vs. 82ns)[6]. In contrast, increased NVM read latency has negligible impact on one-sided RDMA READ since the PCIe latency is the dominant factor (~1000ns [35]).

Unlike *remote read*, the performance of *remote write* is much slower than that of DRAM, as shown in Figure 6 and Figure 7. More importantly, these results are not optimal because the measured performance is far from the theoretical limit of NVM or RDMA. For example, one-sided RDMA WRITE can achieve only 29% of the NVM peak write throughput (15M vs. 52M reqs/sec[7]) for small 16B writes.

[6]Measured by Intel Memory Latency Checker (MLC) [17].

[7]We estimate the NVM peak write throughput by dividing its peak write

Table 3: *A summary of design advice, optimization hints, and whether the hints can apply to a specific RDMA primitive. ✓ indicates a positive optimization effect, and "–" means the hint does not target the case.*

Design advice	Optimization hints	One-sided	Two-sided
A1. Configuration (§4.1)	H1. Avoid cross-socket NVM accesses	✓	✓
	H2. Limit concurrent access to a single NVM DIMM for two-sided RDMA	-	✓
	H3. Disable DDIO; if DDIO must be enabled, use two-sided RDMA	✓	-
A2. Access pattern (§4.2)	H4. Use `ntstore` instead of `store` for large writes	-	✓
	H5. Use XPLine granularity for writes	✓	✓
	H6. Use PCIe DW granularity (64B) for small writes (i.e., less than XPLine)	✓	-
	H7. Use cacheline granularity (64B) with `ntstore` for small writes (i.e., less than XPLine)	-	✓
	H8. Use less atomic operations on NVM	✓	✓
A3. RDMA-aware (§4.3)	H9. Enable outstanding request with doorbell batching for one-sided persistent WRITE	✓	-

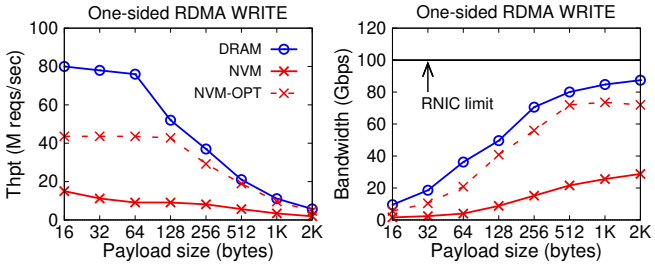

Figure 6. *A comparison of one-sided RDMA WRITE performance on DRAM and NVM, (a) throughput and (b) aggregated bandwidth. NVM-OPT applies the optimizations from §4.*

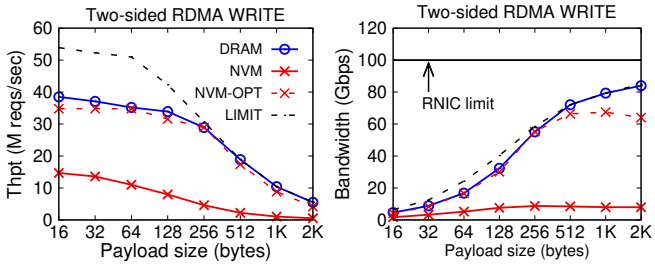

Figure 7. *A comparison of two-sided RDMA WRITE performance on DRAM and NVM, (a) throughput and (b) aggregated bandwidth. LIMIT is measured when the server directly returns to the client without writing the payload. NVM-OPT applies the optimizations from §4.*

Further, one-sided and two-sided RDMA saturate only 38% and 12.5% of the NVM's peak write bandwidth for large 2,048B writes, respectively. Therefore, there is significant room for improvement.

Our approach. To understand why *remote write* has inferior performance, we conduct a systematic study to summarize various performance-relevant factors for RDMA to write NVM. Inspired by a recent CPU-specific NVM study [59], we mainly follow two directions. First, we investigate what system configurations can affect RDMA's efficiency with NVM (§4.1). Second, we study which access patterns from RDMA are friendly to NVM (§4.2). For each direction, we empirically study whether known NVM optimizations are necessary or optimal for RDMA. The results show that some setups are not necessary, while some optimizations are sub-optimal for RDMA. To this end, we present new optimizations by fully considering NVM characteristics with RDMA. Finally, we use existing RDMA optimizations to improve the persistent write of one-sided RDMA atop of NVM (§4.3).

bandwidth (100Gbps) with the size of XPLine (256B).

A preview of the optimization results. Figure 6 and Figure 7 present our optimized version of one-sided and two-sided RDMA NVM write (*NVM-opt*). After applying all the optimizations, they have significantly better performance and achieve close to the NVM limit. For example, one-sided 16B RDMA NVM WRITE achieves 45M reqs/sec, 87% of the ideal peak throughput of NVM write.

4 Design Advice

This section summarizes design advice and optimization hints for high-performance RDMA-NVM systems, as shown in Table 3. We present both new optimizations (e.g., Hint 6, **H6**) and brief descriptions of known optimizations. Further, we show that some known optimizations that only consider CPU accessing NVM are sub-optimal for RDMA (e.g., **H5**). Among these optimizations, **H1–H8** applies to systems that use Optane PM as volatile storage and persistent storage, while **H9** only targets systems that use Optane PM as per-

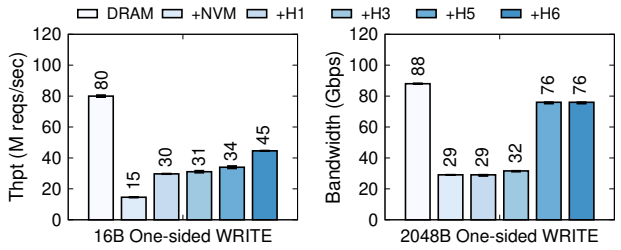

Figure 8. *Factor analysis of the optimizations used to improve NVM write performance atop of one-sided RDMA WRITE under (a) small payload (16B) and (b) large payload (2,048B). We do not include **H8** as it does not target remote write. Note that the error bars are small.*

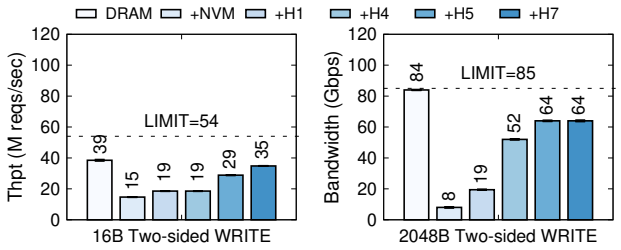

Figure 9. *Factor analysis of the optimizations used to improve NVM write performance atop of two-sided RDMA for (a) small payload (16B) and (b) large payload (2,048B). LIMIT is measured as the server directly returns to the client without writing the payload. We do not include **H8** as it does not target remote write. Note that the error bars are small.*

sistent storage.

We make two assumptions about the hardware components. First, the RNIC is a PCIe-based device, which holds for most existing RNICs [24]. Second, the NVM is Optane PM [36], the first (and only) commercially available NVM device. Unless otherwise stated, we use the same *remote write* microbenchmark in §3 for the study.

4.1 Configuration advice

A prior CPU-specific NVM study [59] has provided valuable configuration setups (e.g., NUMA setup) for the CPU to better utilize NVM. We summarize these setups in **H1** and **H2**. A natural question to answer is: do RDMA-NVM systems require the same setups? Our study reveals that first, RDMA-NVM systems should also consider **H1**. Meanwhile, one-sided RDMA does not necessarily require **H2** as the CPU. Finally, RDMA introduces a new configuration option, **H3**. Succinctly, the configuration advice for RDMA-NVM systems is the following three optimization hints:

H1. Avoid cross-socket NVM accesses;

H2. Limit concurrent access to a single NVM DIMM for two-sided RDMA;

H3. Disable DDIO; If DDIO must be enabled, use two-sided RDMA for large NVM writes;

Hint H1. Yang *et al*. [59] found that the NVM write bandwidth of a socket could be halved from other sockets. Since an RNIC leverages its attached socket to access NVM from another socket (see Figure 1), slow cross-socket NVM accesses also impact RDMA-NVM systems. Figure 9 and Figure 8 illustrate this: removing cross-socket access improves the baseline two-sided and one-sided RDMA write performance by up to 2.4X (8Gbps vs. 19Gbps) and 2X (15M vs. 30M reqs/sec), respectively. Thus, RDMA-NVM systems should also avoid cross-socket NVM accesses.

Apply H1. Readers may wonder whether **H1** is feasible in real systems. One strategy to apply **H1** is first attaching one RNIC for each socket, and then treating each socket as a logical node in a cluster. Such a setup is common in RDMA-capable systems [11, 29, 51, 54, 62], and it naturally avoids cross-socket NVM access. Furthermore, even if there are insufficient RNICs for each socket to have a dedicated RNIC, one can adopt the techniques in IOctopus [42] to apply **H1**. Using IOctopus, one RNIC can simultaneously communicate with multiple sockets. Hence, RDMA can directly access the NVM bypassing the attached socket.

Hint H2. Another observation from Yang *et al*. [59] is that the CPU fails to scale up when writing to a single NVM DIMM. This phenomenon also affects two-sided RDMA because it uses CPU to write to the NVM. As shown in Figure 10(a), compared to using four threads at the server to handle writes, two-sided RDMA-NVM write bandwidth drops 37% when using 20 threads. Note that to avoid interference from other factors, we have enabled all other optimizations hints in this experiment. Therefore, system designers should also reduce concurrent access to a single NVM DIMM for two-sided RDMA. In practice, designers can control which DIMM to access by selecting the appropriate NVM addresses [59]. For example, assuming the starting address of the NVM is 4KB-aligned, and the NVM is interleaved on 6 DIMMS, the first and seventh 4KB is on the first DIMM, and the second 4KB is on the second DIMM, etc.

Does one-sided RDMA suffer from the same issue? To quantify this, we further conduct an experiment to measure the concurrent write performance of one-sided RDMA-NVM WRITE. In this benchmark, we increase the number of clients that send concurrent one-sided RDMA WRITE to a single NVM DIMM, and measure their aggregated bandwidth. Figure 10(b) presents the results: one-sided RDMA WRITE scales well with the increased number of concurrent requests. This implies that one-sided RDMA is more robust when concurrently accessing a single NVM DIMM.

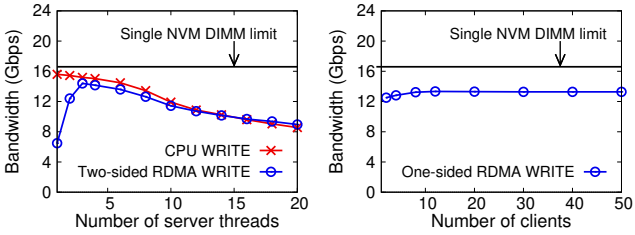

Figure 10. *Effects of concurrent accesses to a single NVM DIMM for (a) two-sided RDMA and (b) one-sided RDMA WRITE using a 2048B payload. The write bandwidth limit of a single NVM DIMM is about 16Gbps (one-sixth of the evaluating Optane PM). Note that we have enabled all other optimizations for both RDMA primitives in this experiment.*

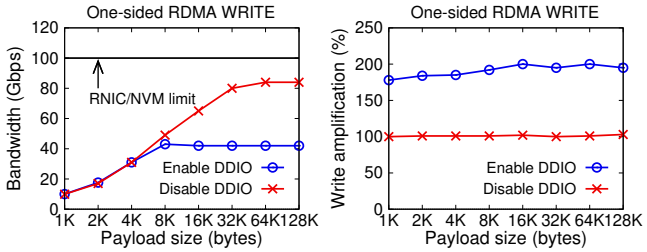

Figure 11. *(a) Effects of DDIO to bulk one-sided RDMA-NVM WRITE, (b) write amplification analysis of one-sided RDMA-NVM WRITE. Note that we have enabled all other optimizations in this experiment.*

To the best of our knowledge, it remains unknown why the CPU cannot scale up when accessing a single NVM DIMM. Yang *et al*. [59] suspected that the high NVM access latency may cause head-of-line blocking effects at the processor. Similarly, we suspect that the RNIC can scale well for one-sided RDMA because the increased latency of NVM access is not that significant compared to PCIe latency.

Hint H3. One important RDMA-specific configuration is whether to enable DDIO, which controls the destination of one-sided RDMA WRITE (§2.3). A prior study has revealed that DDIO has a huge performance impact on one-sided RDMA-NVM WRITE for large payloads [22]; we also made a similar observation during our study. Consequently, **H3** is an important configuration setup for RDMA-NVM systems.

Impact of DDIO. Figure 11(a) shows the effects of DDIO on one-sided RDMA-NVM WRITE. Since large RDMA-NVM WRITE is more sensitive to DDIO, we use a bulk write benchmark where a single client issues a sufficient large payload to measure the peak bandwidth of WRITE. With DDIO enabled, we observe that WRITE can only reach half of the NVM peak bandwidth (42Gbps vs. 100Gbps). On the other hand, the WRITE can achieve close to NVM peak bandwidth with DDIO disabled.

Ideally, the client should saturate the RNIC(NVM) bandwidth in this benchmark. However, it fails because DDIO changes the sequential writes from RNIC to random writes to NVM. Figure 12(a) illustrates this: the RNIC first sequentially writes the data into the cache, after that the cache randomly evicts the data to the NVM. Random writes cannot saturate NVM bandwidth because NVM has less chance to merge adjacent writes to avoid write amplification (see §2.1).

Measuring the write amplification of DDIO. To quantify the effect of DDIO, Figure 11(b) analyzes the write amplification of NVM[8] with different configurations. We can see that enabling DDIO incurs a roughly 2X write amplification to NVM WRITE, which explains why the bandwidth is halved for a large payload (e.g., more than 64KB).

Implications for RDMA-NVM systems. If the systems must use one-sided RDMA WRITE to saturate the NVM bandwidth, we recommend considering **H3** to turn off the DDIO first. The system can statically enable/disable DDIO via the BIOS setups [58] or dynamically adjust the bits in Integrated I/O (IIO) Configuration Registers [1, 13, 18] at runtime. We follow prior work [13] to use configuration registers to configure DDIO at runtime.

Limitations of disabling DDIO. On the current hardware platform, the DDIO configuration affects all the devices on a processor. Thus, server CPU will have a poorer cache locality for DMA-ed data (e.g., messages in two-sided primitives) with DDIO disabled, resulting in degraded two-sided RDMA performance. For example, we measured a 57% peak throughput drop (54M vs. 23M reqs/sec) of two-sided RDMA primitives after disabling DDIO. Therefore, the current RDMA-NVM systems will make a trade-off between the bandwidth of one-sided RDMA NVM WRITE and the performance of two-sided RDMA. Considering the limitations of disabling DDIO, we extend **H3** as follows:

H3 (extended). If DDIO must be enabled, use two-sided RDMA for large NVM writes;

Two-sided RDMA can leverage the CPU at the server to fully utilize NVM [59]. Hence, RDMA-NVM systems can adopt a hybrid approach for implementing NVM write based on the request payload size.

4.2 Access pattern advice

Tuning systems NVM access pattern according to the Optane PM's features is critical to NVM-aware systems. Known optimizations for CPU include choosing the appropriate CPU instructions and using the proper access granularity [59]. We summarize these hints in **H4–H5**:

[8] We measure the write amplification of DDIO via NVM counters. §2.1 describes the measurements in more detail.

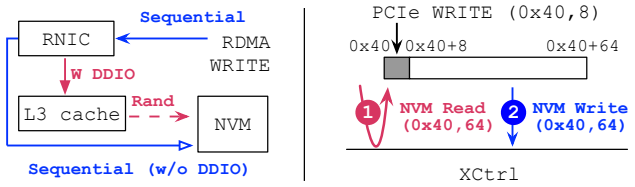

Figure 12. *(a) DDIO changes the sequential accesses of RNIC into random accesses. (b) An example of PCIe partial write: PCIe sends two NVM requests when writing 8B at address 0x40.*

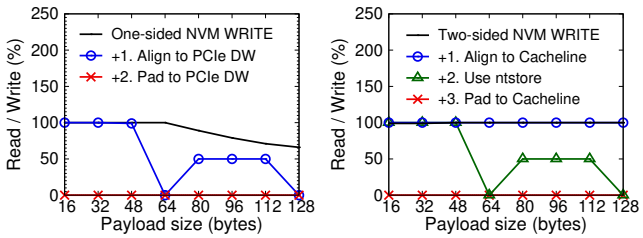

Figure 13. *The ratio of extra NVM read per write of (a) one-sided RDMA WRITE and (b) two-sided RDMA on the remote write benchmark. Aligning and padding the write payload to device access granularity (e.g., PCIe DW) is sufficient to avoid unnecessary NVM reads for one-sided RDMA WRITE. On the other hand, two-sided RDMA further needs to use* `ntstore`.

H4. Use `ntstore` instead of `store` for large writes;

H5. Use XPLine granularity for writes;

Both hints can also benefit RDMA-NVM systems. However, we found **H5** is not optimal when considering RDMA, especially for small writes, because it incurs huge network amplification. For example, applying **H5** only improves the performance of 16B one-sided RDMA NVM WRITE by 1.1X (31M vs. 34M reqs/sec), which remains far from NVM's ideal processing rate (52M reqs/sec). In this case, **H5** will incur 16X (256B vs. 16B) network amplification to one-sided RDMA WRITE. Since moving 256B data to DRAM over RDMA is even slower than NVM ideal processing rate (37M vs. 52M reqs/sec, as shown in Figure 6), the network would first become the bottleneck for small writes.

To this end, we found the key for small writes to fully utilize NVM is to *avoid sending unnecessary read requests to the NVM*. As we have mentioned in §2.1, NVM read requests compete for **XCtrl** processing power with NVM write requests. Hence, unnecessary read requests would drastically reduce the NVM write throughput. This fact allows using a smaller access granularity to saturate NVM's processing rate with RDMA.

CPU and RNIC generate an extra read request to NVM if the payload of the write request does not fit their access granularities (i.e., cacheline and PCIe DW). They execute such a write request in a read-modify-write pattern to avoid overwriting the original content. Thus, using the CPU/RNIC access granularity is sufficient to prevent sending unnecessary reads from the device to NVM. Based on this observation, we first propose **H6–H7** to complement **H5** for small writes. Further, since the read-modify-write pattern is not friendly to NVM, we also propose **H8** to suggest systems use fewer such operations. Specifically, we propose the following hints to complement **H4–H5**:

H6. For one-sided RDMA, use PCIe data word (64B) granularity when the payload is smaller than XPLine;

H7. For two-sided RDMA, use cacheline granularity (64B) with `ntstore` when the payload is smaller than XPLine;

H8. Use less atomic operations on NVM;

Hint H6. PCIe issues write in a read-modify-write pattern with *PCIe partial-write*. Figure 12(b) shows a concrete example where the RNIC uses PCIe to write 8B at 0x40. It will first send a read request to the NVM to read the data word at 0x40 (❶). Then, it overwrites the entire data word according to the write request (❷).

Figure 8(a) presents the optimized performance of **H6** to one-sided RDMA WRITE: it improves the performance of 16B WRITE to 87% of the NVM's peak write throughput (45M vs. 52M reqs/sec). The result is 1.3X faster than applying **H5** thanks to the reduced network amplification. Figure 13(a) further examines how eliminating PCIe partial write helps to prevent sending read requests to the NVM. We apply **H6** by first aligning the written address to PCIe DW (+1. Align to PCIe DW), and then padding the payload size (+2. Pad to PCIe DW) to a multiple of PCIe DW. As we can see, after applying both steps, one-sided RDMA WRITE does not issue a read request to the NVM. Consequently, small WRITEs (e.g., no larger than 64B) can reach close to the NVM processing limit.

We should mention that reducing the PCIe partial write may also benefit one-sided RDMA-DRAM WRITE. However, our experiments show that it may even have a negative effect on DRAM WRITE. For example, the 16B DRAM WRITE has a 25% performance degradation on our testbed after applying **H6**.

Hint H7. Similar with **H6**, **H7** suggests how to prevent read-modify-write for two-sided RDMA. As shown in Figure 8, it improves the 16B two-sided RDMA-NVM WRITE performance by 1.8X (19M vs. 35M reqs/sec). To apply **H7**, two-sided RDMA should use `ntstore` together with use cacheline granularity (+1. Align to Cacheline and +3. Pad to Cacheline, as shown in Figure 13(b)). This is because, the CPU would pre-fetch the cacheline using NVM read with

`store`. As shown in Figure 13(b), two-sided RDMA-NVM WRITE always achieves a 100% read/write ratio even after using the proper access granularity.

Hint H8. A lesson learned from **H7–H8** is that the read-modify-write access pattern is not friendly to NVM. Atomic operations (e.g., one-sided RDMA ATOMIC compare and swap) naturally follow a read-modify-write pattern. Worse, the designer cannot apply prior hints to optimize atomic operations. For instance, one-sided RDMA ATOMICs cannot apply **H6** since they use a fixed 8B granularity. Consequently, we suggest using fewer atomics on NVM for RDMA. Note that we are not suggesting disabling atomics, but *moving the data for atomic operations (e.g., spinlock) from NVM to DRAM whenever possible*.

Discussion of H6–H8. Although applying **H6** and **H7** may waste NVM storage for storing the padding, while adopting **H8** could change the persistent semantic of atomic data, we believe **H6–H8** is actionable in real systems. This is because there are many scenarios in RDMA-NVM systems that can use **H6-H8** without wasting storage or changing the application semantics. For instance, existing RDMA-NVM enabled databases [10, 11, 25, 53, 54] do not require the lock to be persistent. Thus, we can safely move their lock metadata from NVM to DRAM. Furthermore, distributed logging [10, 11] naturally uses padding to accommodate future logs. Thus, applying **H6** to logging does not introduce additional storage overhead. Finally, as we will present in §6, **H6–H8** can have huge performance improvements for existing systems. For example, **H6–H8** can improve the performance of DrTM+H on the SmallBank [45] benchmark by 1.79X (3.9M vs. 7.0M txns/sec, see Figure 16).

4.3 RDMA-aware advice

Finally, we discuss how known RDMA-aware optimizations can mitigate the inefficiency of implementing persistent write atop of existing hardware platforms. As we have mentioned in the introduction, a strawman approach to implementing persistent write using one-sided RDMA requires two network roundtrips: the first WRITE attempts to store the data to NVM, while the second READ ensures that the written data is flushed to the persistent domain (e.g., Optane PM). Fortunately, it is possible to leverage well-known RDMA-aware optimizations to avoid the additional network roundtrip of READ. **H9** summarizes this fact:

H9. Enable outstanding request with doorbell batching for one-sided persistent RDMA WRITE.

Specifically, outstanding request [23] allows us using the completion of READ as the completion of the WRITE, as long as the two requests are sent to the same QP. Since the READ to persist the WRITE must be post to the same QP as the WRITE (§2.3), we no longer need to wait for the first WRITE to complete. Thus, this optimization reduces the wait time of the first network roundtrip. Applying outstanding request to persistent WRITE is correct because first, later READ flushes previously WRITE [19], and RNIC processes requests from the same QP in a FIFO order [6].

Based on outstanding request, doorbell batching [24] further allows us to send the READ and WRITE in one request using the more CPU and bandwidth efficient DMA, reducing the latency of posting RDMA requests.

On our testbed, a single one-sided RDMA request takes 2μs. Thus, a strawman implementation of *remote persistent write* uses 4μs. After applying **H9**, one-sided *remote persistent write* takes 3μs latency to finish.

5 Discussion of Future Trends

Generality of the study. Our study focuses on specific RNIC (Mellanox ConnectX-5) and NVM (Intel Optane DC Persistent Memory), while other hardware devices may yield different results. Nevertheless, we believe ConnectX-5 is a representative RNIC, as recent generations of Mellanox RNICs (e.g., Connect-IB, ConnectX-4) all share the same architecture. Moreover, Optane PM is the only commercially available NVM. Finally, we also provide open-source tools that the developers can use to examine their design choices under different hardware configurations.

Next-generation NVM. The next-generation of NVM not only has a better performance but also provides a larger scope of persistent domain. First, it will have a 25% higher bandwidth [21]. Second, it will include the processor cache in its persistent domain [4]. This feature is desirable for one-sided RDMA since one-sided RDMA no longer depends on DDIO for WRITE to be persistent (§2.3). Consequently, the designer does not need to make a trade-off between one-sided persistence and two-sided RDMA performance (§4.1).

On the other hand, the new features of next-generation NVM are unlikely to change the advice of our study. First, the primary focus of our study is *how to avoid NVM write becoming the bottleneck* in RDMA-NVM systems (§3), even when NVM write has a comparable performance with RDMA (see Figure 1). Thus, the 25% bandwidth improvement of the next-generation NVM is insufficient to twist the performance comparisons between RDMA and NVM, since future generations of RDMA will have much higher bandwidth. For example, RNIC with 200Gbps bandwidth has already been commercially available [3], which is 2X higher than our evaluated RNIC. Besides performance, the enhanced functionality of next-generation NVM, i.e., putting cache in the persistent domain, is also unlikely to address the current performance issue caused by the cache. This is because the random cache eviction is still not suitable for

NVM. Meanwhile, an extra one-sided RDMA READ is still required to ensure persistence as long as the RNIC is not redesigned.

Suggestions to hardware designers. There are proposals to extend RDMA to cooperate with NVM, e.g., Talpey *et al*. [44] proposed to add a *one-sided commit* primitive to support one-side persistent RDMA WRITE. Nevertheless, our study reveals that existing proposals are insufficient: *the hardware designers not only need to consider hardware extensions to support more functionality, but also should consider extensions for better performance*. For instance, adding an RDMA-version of `ntstore`, e.g., one-sided *non-temporal* RDMA WRITE that allows the WRITE to bypass the cache—can greatly improve the flexibility in configuring DDIO for RDMA-NVM systems (§4.1).

6 Improved System Designs

Existing or future RDMA-NVM systems can benefit from the summarized optimization hints in our study (§4). In this section, we present how we use these hints to improve the performance of two open-source RDMA-NVM systems, a distributed database (DrTM+H [53]), and a distributed file system (Octopus [32]). Both systems are designed when no production NVM is available.

6.1 Distributed database

DrTM+H [53] is a distributed transactional system designed for RDMA and NVM. It fully leverages the power of one-sided and two-sided RDMA to boost transaction execution. We choose it for optimization for two reasons. First, its concurrency control and replication protocol use RDMA and NVM heavily. Therefore, there may exist a huge space for improvements. Second, most existing RDMA-NVM distributed databases adopt a similar protocol [8, 10, 11, 25, 37] as DrTM+H. Thus, our improved DrTM+H design can potentially benefit these systems.

Overview. DrTM+H uses optimistic concurrency control [28] (OCC) to ensure strict serializability and primary-backup replication to achieve high availability [8, 11]. It organizes NVM as a distributed shared memory pool similar to prior work [10, 32, 54], which stores the database records and transaction logs. DrTM+H uses four phases to execute a transaction, each interacts with NVM using RDMA as follows: *Execution* uses one-sided RDMA READ to read records stored in NVM; *Validation* uses one-sided RDMA Compare and swap (CAS) to acquire the lock co-located with the record; *Logging* uses one-sided RDMA WRITE to replicate the transaction updates to backups; *Commit* uses two-sided RDMA to update and unlock the records.

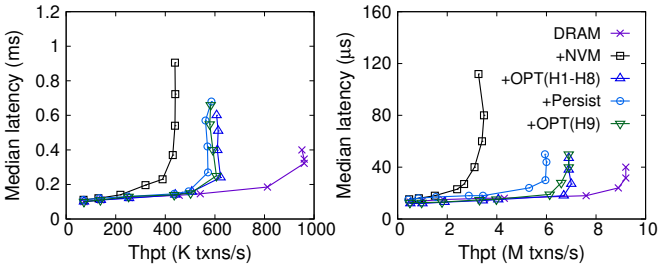

Figure 14. *The performance of DrTM+H on (a) TPC-C/no and (b) SmallBank.*

6.1.1 Optimizations

The original DrTM+H neither considers the performance features of NVM (§2.1), nor the persistence issue of one-sided RDMA WRITE (§2.3). In this section, we first apply optimizations that are beneficial to DrTM+H both when using NVM to extend DRAM capacity and to support durability (i.e., **H1–H8**):

- Separate the memory pool from different sockets to avoid cross-socket NVM access (**H1**).
- Configure DrTM+H with DDIO disabled (**H3**).
- Use `ntstore` to optimize the *commit phase* (**H4**).
- Align and pad logs/records larger than 256B to XPLine granularity (**H5**).
- Align and pad logs/records smaller than 256B to 64B granularity (**H6** + **H7**).
- Implement a DRAM-based lock service for the *validation phase* (**H8**). Note that it is safe not persisting the locks in NVM because DrTM+H does not require the locks to be persistent even when durability is enabled.

Second, we use **H9** to optimize DrTM+H when use NVM to support durability. When following existing approach [20] to support durable transactions, DrTM+H has to use two network roundtrips to persist a single transaction log at the logging phase. With the help of **H9**, the logging phase only use one roundtrip:

- Implement remote persistent log with **H9** in one roundtrip.

6.1.2 Evaluation

Setup. By default, DrTM+H executes transactions in a symmetric setting [10]: each machine both stores database partition and executes transactions. Nevertheless, we evaluate it in an asymmetric setting due to the lack of NVM-capable machines (Table 2): the NVM server stores the

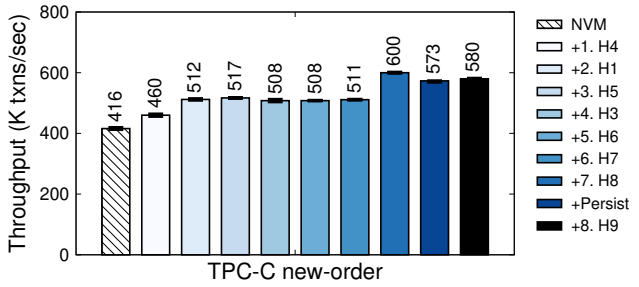

Figure 15. *The contribution of optimizations to the throughput of DrTM+H for TPC-C/no. Note that the error bars are small.*

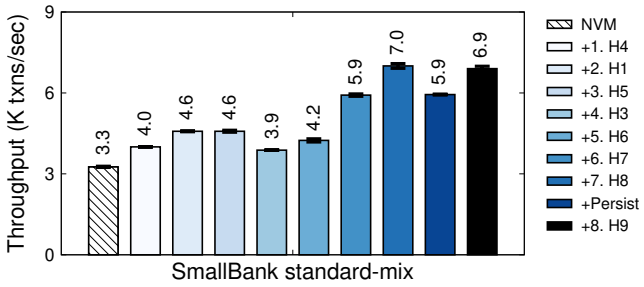

Figure 16. *The contribution of optimizations to the throughput of DrTM+H for SmallBank. Note that the error bars are small.*

database while other machines execute transactions. We use two representative OLTP benchmarks to evaluate its performance:

TPC-C/no [46] simulates the workload of an ordering system that contains complex read/write workloads. We configure the NVM server to store ten warehouses and other machines to execute the *new-order* (no) transaction, the dominant transaction of TPC-C [46].

SmallBank [45] models a simple banking application where each transaction issues one or two read/write requests. We store 20,000,000 bank accounts at the NVM server and run the standard-mix transactions at clients.

Comparing targets. In Figure 14, **DRAM** is the vanilla DrTM+H running on DRAM. **+NVM** runs DrTM+H on Optane PM, and **+OPT(H1-H8)** further applies optimizations (§6.1.1). **+Persist** adopts an existing approach [20] to support durability atop of **+OPT(H1-H8)**. Finally, **+OPT(H9)** optimizes **+Persist** with **H9**.

Performance without Persistence. Figure 14 presents the throughput-latency results of both workloads. We plot the graph by increasing the number of clients until the throughput is saturated. **+OPT(H1-H8)** improves the DrTM+H's performance under TPC-C/no and SmallBank by 1.45X and 2.20X, respectively.

To analyze the contributions of each optimization, Figure 15 and Figure 16 further present factor analyses of evaluation results on TPC-C/no and SmallBank, respectively. First, we can see that several hints are beneficial to both workloads. For example, **H8** (use atomic operations less on NVM) speedups TPC-C/no and SmallBank by 1.17X and 1.19X, respectively. On the other hand, some hints have negative effects for certain workloads: SmallBank drops 15% throughput when adding **H3**, this is because **H3** is only beneficial when the application is bottlenecked by NVM's bandwidth. Finally, some hints have more contributions to SmallBank than TPC-C/no. For example, **H7** has a 1.4X speedup on SmallBank but does not affect TPC-C/no. **H7** only improves transaction utilizations of NVM write throughput, while SmallBank is more sensitive to the NVM write throughput utilization due to its simpler workloads.

Performance with persistence. As shown in Figure 14, supporting persistent transaction (**+Persist**) adds 5% and 15% performance overhead to **+OPT(H1-H8)** on TPC-C/no and SmallBank, respectively. The overhead is from the additional one-sided RDMA READ at the logging phase. Hence, reducing this network roundtrip with **H9** improves TPC-C/no and SmallBank's performance by 1.01X and 1.17X, respectively.

6.2 Distributed file system

Octopus [32] is a distributed file system designed for RDMA and NVM. Due to space limitations, we only give a brief overview of it and our applied optimizations.

Overview. Octopus uses a distributed NVM pool to store the file system metadata and its file data blocks. It achieves high throughput and bandwidth for reading/write file data through *Client-Active Data I/O*: the client directly read-/write a file's data block using one-sided RDMA READ-/WRITE. Besides *Client-Active Data I/O*, Octopus also leverages RDMA-enabled distributed transactions to update the filesystem metadata. Similar to DrTM+H (§6.1), its transactional protocol uses one-sided RDMA ATOMICs to coordinate conflicting metadata operations.

Optimizations. We focus on improving Octopus's *Client-Active Data I/O* because the distributed transaction is not supported in its current public available codebase[9]. Nevertheless, we believe our findings can also improve distributed transaction performance in Octopus, e.g., using **H8** to improve its lock performance.

6.2.1 Evaluation

We use the same Data I/O benchmark in the Octopus paper [32] for the evaluation. In this benchmark, each client writes a fixed payload to a random location in a randomly chosen file. The client first uses two-sided RDMA to query

[9] https://github.com/thustorage/octopus

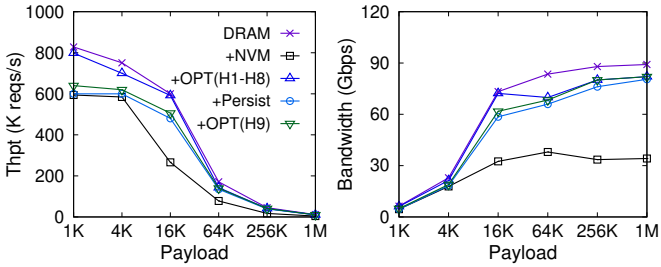

Figure 17. *Data I/O (a) throughput and (b) bandwidth of Octopus (Multiple Clients).*

the file metadata (e.g., data block addresses). Then, it writes the data payload with one-sided RDMA WRITE.

Comparing targets. In Figure 17, **DRAM** is the vanilla Octopus using DRAM to emulate the NVM pool. **+NVM** uses Optane PM as NVM pool, and **+OPT(H1-H8)** applies our optimizations to **+NVM**. **+Persist** further adopts an existing approach [20] to support synchronous durable file write atop pf **+OPT(H1-H8)**. Finally, **+OPT(H9)** leverages **H9** to reduce network roundtrips for persistence of **+Persist**.

Performance. As shown in Figure 17, **+OPT(H1-H8)** improve **+NVM** by up to 2.4X (from 1.2X), mainly due to applying **H3**. Without **H3**, Octopus's client-active I/O cannot fully saturate NVM's write bandwidth because it uses one-sided RDMA WRITE with DDIO enabled to write to the NVM. Further, **+OPT(H9)** outperforms **+Persist** by 1.06X (from 1.02X), thanks to the reduced RDMA roundtrips for persistent write.

7 Related Work

RDMA-NVM systems. We continue the line of research of using RDMA and NVM to improve the performance and reliability of distributed systems [8, 11, 32, 33, 38, 39, 43, 58, 61]. Kashyap *et al*. [27] explores the trade-offs of using different methods to ensure NVM write persistence with RDMA. They conduct their experiments on emulated NVM. Our study instead focuses on Optane PM. Orion [58] is a distributed file system designed for RDMA and NVM. It does not consider RDMA-ware optimizations (i.e., **H9**) and chooses two-sided RDMA for the persistent write. Our study provides another design decision for them. Hotpot [38] uses RDMA and NVM to build a distributed persistent shared memory. AsymNVM [33] proposes an asymmetric architecture to use NVM with RDMA. Though AsymNVM is evaluated with Optane PM, it does not consider the performance features of Optane PM. Hence, we believe our study can further improve its performance on Optane PM.

RDMA-aware optimizations. Our work is built upon broadly explored RDMA-aware optimizations [7, 10, 24, 25, 47, 52, 53]. The evaluating execution framework [53] has integrated most of these optimizations. FaRM [10] proposes various techniques to utilize RNIC's cache better, e.g., using huge pages. Kalia *et al*. [24] present the importance of understanding the low-level factors of how RNIC works. Based on this, they propose various RDMA-aware optimizations such as doorbell batching. FaSST [25] presents an efficient and scalable RPC framework atop two-sided RDMA datagram primitive. LITE [47] uses kernel indirection to improve the scalability of one-sided RDMA. Wukong [40, 56, 60] leverages RDMA to improve the performance of distributed graph store and further considers the interaction between RDMA and GPU [51].

NVM-aware systems. Except for RDMA-NVM systems, researchers are building NVM-aware systems for decades, including but not limited to file systems [9,12,57], NVM-aware data structures [48, 63], key-value stores [26], NVM-aware JVM [41, 55], and transactions on NVM [14, 15, 30, 34, 50]. Like RDMA-NVM systems, most of them use emulated NVM since there is no commercially available NVM at that time. We hope our study can further inspire future research on revisiting previous NVM-aware systems on Optane PM.

8 Conclusion

Designing high-performance RDMA-NVM systems requires a clear understanding of the interaction between RDMA and NVM. This paper provides a systematic study on how to best utilize NVM with RDMA, which summarizes nine optimization hints. By examining existing RDMA-NVM systems with these hints, we found room for improvements, especially for those not targeting production NVM: our optimized DrTM+H is up to 2.2X faster on Optane PM, while our optimized Octopus file system is up to 2.4X faster. We believe our summarized hints as well as experiences in applying them to existing systems can benefit future systems developers when designing systems with RDMA and NVM.

9 Acknowledgment

We sincerely thank our shepherd Aleksandar Dragojevic and the anonymous reviewers for their insightful comments and feedback. This work was supported in part by the National Key Research & Development Program of China (No. 2020YFB2104100), the National Natural Science Foundation of China (No. 61732010, 61925206), and a grant from Huawei Technologies. Corresponding author: Rong Chen (rongchen@sjtu.edu.cn).

References

[1] NetCAT. https://www.vusec.net/projects/netcat/.

[2] The Volatile Benefit of Persistent Memory. https://memcached.org/blog/persistent-memory/, 2020.

[3] 200Gb/s ConnectX-6 Ethernet Single/Dual-Port Adapter IC. https://www.mellanox.com/products/ethernet-adapter-ic/connectx-6-en-ic, 2021.

[4] Build Persistent Memory Applications with Reliability Availability and Serviceability. https://software.intel.com/content/www/us/en/develop/articles/build-pmem-apps-with-ras.html, 2021.

[5] ipmctl. https://github.com/intel/ipmctl, 2021.

[6] ASSOCIATION., I. T. Infiniband architecture specification. https://cw.infinibandta.org/document/dl/7859, 2015.

[7] CHEN, H., CHEN, R., WEI, X., SHI, J., CHEN, Y., WANG, Z., ZANG, B., AND GUAN, H. Fast in-memory transaction processing using rdma and htm. *ACM Trans. Comput. Syst. 35*, 1 (July 2017).

[8] CHEN, Y., WEI, X., SHI, J., CHEN, R., AND CHEN, H. Fast and general distributed transactions using rdma and htm. In *Proceedings of the Eleventh European Conference on Computer Systems* (New York, NY, USA, 2016), EuroSys '16, Association for Computing Machinery.

[9] DONG, M., AND CHEN, H. Soft updates made simple and fast on non-volatile memory. In *2017 USENIX Annual Technical Conference (USENIX ATC 17)* (Santa Clara, CA, July 2017), USENIX Association, pp. 719–731.

[10] DRAGOJEVIĆ, A., NARAYANAN, D., CASTRO, M., AND HODSON, O. Farm: Fast remote memory. In *11th USENIX Symposium on Networked Systems Design and Implementation (NSDI 14)* (Seattle, WA, Apr. 2014), USENIX Association, pp. 401–414.

[11] DRAGOJEVIĆ, A., NARAYANAN, D., NIGHTINGALE, E. B., RENZELMANN, M., SHAMIS, A., BADAM, A., AND CASTRO, M. No compromises: Distributed transactions with consistency, availability, and performance. In *Proceedings of the 25th Symposium on Operating Systems Principles* (New York, NY, USA, 2015), SOSP '15, Association for Computing Machinery, p. 54–70.

[12] DULLOOR, S. R., KUMAR, S., KESHAVAMURTHY, A., LANTZ, P., REDDY, D., SANKARAN, R., AND JACKSON, J. System software for persistent memory. In *Proceedings of the Ninth European Conference on Computer Systems* (New York, NY, USA, 2014), EuroSys '14, Association for Computing Machinery.

[13] FARSHIN, A., ROOZBEH, A., JR., G. Q. M., AND KOSTIĆ, D. Reexamining direct cache access to optimize i/o intensive applications for multi-hundred-gigabit networks. In *2020 USENIX Annual Technical Conference (USENIX ATC 20)* (July 2020), USENIX Association, pp. 673–689.

[14] HSU, T. C.-H., BRÜGNER, H., ROY, I., KEETON, K., AND EUGSTER, P. Nvthreads: Practical persistence for multi-threaded applications. In *Proceedings of the Twelfth European Conference on Computer Systems* (New York, NY, USA, 2017), EuroSys '17, Association for Computing Machinery, p. 468–482.

[15] HU, Q., REN, J., BADAM, A., SHU, J., AND MOSCIBRODA, T. Log-structured non-volatile main memory. In *2017 USENIX Annual Technical Conference (USENIX ATC 17)* (Santa Clara, CA, July 2017), USENIX Association, pp. 703–717.

[16] INTEL. Intel® data direct i/o technology. https://www.intel.com/content/www/us/en/io/data-direct-i-o-technology.html, 2019.

[17] INTEL. Intel® memory latency checker v3.7. https://software.intel.com/en-us/articles/intelr-memory-latency-checker, 2019.

[18] INTEL. Intel® xeon® processor e5 v4 product family. https://www.intel.com/content/www/us/en/processors/xeon/xeon-e5-v4-datasheet-vol-2.html, 2019.

[19] INTEL. Persistent Memory Replication Over Traditional RDMA. https://software.intel.com/en-us/articles/persistent-memory-replication-over-traditional-rdma-part-1-understanding-remote-persistent, 2019.

[20] INTEL. The librpmem library. https://pmem.io/pmdk/librpmem/, 2019.

[21] INTEL. Intel® Optane persistent memory 200 series. https://www.intel.com/content/www/us/en/products/docs/memory-storage/optane-persistent-memory/optane-persistent-memory-200-series-brief.html, 2020.

[22] KALIA, A., ANDERSEN, D., AND KAMINSKY, M. Challenges and solutions for fast remote persistent memory access. In *Proceedings of the 11th ACM Symposium on Cloud Computing* (New York, NY, USA, 2020), SoCC '20, Association for Computing Machinery, p. 105–119.

[23] KALIA, A., KAMINSKY, M., AND ANDERSEN, D. G. Using RDMA efficiently for key-value services. In *Proceedings of the 2014 ACM Conference on SIGCOMM* (New York, NY, USA, 2014), SIGCOMM '14, Association for Computing Machinery, p. 295–306.

[24] KALIA, A., KAMINSKY, M., AND ANDERSEN, D. G. Design guidelines for high performance RDMA systems. In *2016 USENIX Annual Technical Conference (USENIX ATC 16)* (Denver, CO, June 2016), USENIX Association, pp. 437–450.

[25] KALIA, A., KAMINSKY, M., AND ANDERSEN, D. G. Fasst: Fast, scalable and simple distributed transactions with two-sided (RDMA) datagram rpcs. In *12th USENIX Symposium on Operating Systems Design and Implementation (OSDI 16)* (Savannah, GA, Nov. 2016), USENIX Association, pp. 185–201.

[26] KANNAN, S., BHAT, N., GAVRILOVSKA, A., ARPACI-DUSSEAU, A., AND ARPACI-DUSSEAU, R. Redesigning LSMs for nonvolatile memory with NoveLSM. In *2018 USENIX Annual Technical Conference (USENIX ATC 18)* (Boston, MA, July 2018), USENIX Association, pp. 993–1005.

[27] KASHYAP, S., QIN, D., BYAN, S., MARATHE, V. J., AND NALLI, S. Correct, fast remote persistence. *arXiv preprint arXiv:1909.02092* (2019).

[28] KUNG, H. T., AND ROBINSON, J. T. On optimistic methods for concurrency control. *ACM Trans. Database Syst. 6*, 2 (June 1981), 213–226.

[29] LIM, H., HAN, D., ANDERSEN, D. G., AND KAMINSKY, M. MICA: A holistic approach to fast in-memory key-value storage. In *11th USENIX Symposium on Networked Systems Design and Implementation (NSDI 14)* (Seattle, WA, Apr. 2014), USENIX Association, pp. 429–444.

[30] LIU, M., ZHANG, M., CHEN, K., QIAN, X., WU, Y., ZHENG, W., AND REN, J. DudeTM: Building durable transactions with decoupling for persistent memory. In *Proceedings of the Twenty-Second International Conference on Architectural Support for Programming Languages and Operating Systems* (New York, NY, USA, 2017), ASPLOS '17, Association for Computing Machinery, p. 329–343.

[31] LIU, X., HUA, Y., LI, X., AND LIU, Q. Write-optimized and consistent RDMA-based NVM systems. *arXiv preprint arXiv:1906.08173* (2019).

[32] LU, Y., SHU, J., CHEN, Y., AND LI, T. Octopus: an RDMA-enabled distributed persistent memory file system. In *2017 USENIX Annual Technical Conference (USENIX ATC 17)* (Santa Clara, CA, July 2017), USENIX Association, pp. 773–785.

[33] MA, T., ZHANG, M., CHEN, K., SONG, Z., WU, Y., AND QIAN, X. Asymnvm: An efficient framework for implementing persistent data structures on asymmetric NVM architecture. In *Proceedings of the Twenty-Fifth International Conference on Architectural Support for Programming Languages and Operating Systems* (New York, NY, USA, 2020), ASPLOS '20, Association for Computing Machinery, p. 757–773.

[34] MEMARIPOUR, A., BADAM, A., PHANISHAYEE, A., ZHOU, Y., ALAGAPPAN, R., STRAUSS, K., AND SWANSON, S. Atomic in-place updates for non-volatile main memories with Kamino-Tx. In *Proceedings of the Twelfth European Conference on Computer Systems* (New York, NY, USA, 2017), EuroSys '17, Association for Computing Machinery, p. 499–512.

[35] NEUGEBAUER, R., ANTICHI, G., ZAZO, J. F., AUDZEVICH, Y., LÓPEZ-BUEDO, S., AND MOORE, A. W. Understanding PCIe performance for end host networking. In *Proceedings of the 2018 Conference of the ACM Special Interest Group on Data Communication* (New York, NY, USA, 2018), SIGCOMM '18, Association for Computing Machinery, p. 327–341.

[36] RYAN SMITH. Intel Announces Optane Storage Brand For 3D XPoint Products. https://www.anandtech.com/show/9541/intel-announces-optane-storage-brand-for-3d-xpoint-products, 2015.

[37] SHAMIS, A., RENZELMANN, M., NOVAKOVIC, S., CHATZOPOULOS, G., DRAGOJEVIĆ, A., NARAYANAN, D., AND CASTRO, M. Fast general distributed transactions with opacity. In *Proceedings of the 2019 International Conference on Management of Data* (New York, NY, USA, 2019), SIGMOD '19, Association for Computing Machinery, p. 433–448.

[38] SHAN, Y., TSAI, S.-Y., AND ZHANG, Y. Distributed shared persistent memory. In *Proceedings of the 2017 Symposium on Cloud Computing* (New York, NY, USA, 2017), SoCC '17, Association for Computing Machinery, p. 323–337.

[39] SHEN, S., CHEN, R., CHEN, H., AND ZANG, B. Retrofitting High Availability Mechanism to Tame Hybrid Transaction/-Analytical Processing. In *15th USENIX Symposium on Operating Systems Design and Implementation (OSDI 21)* (July 2021), USENIX Association.

[40] SHI, J., YAO, Y., CHEN, R., CHEN, H., AND LI, F. Fast and concurrent RDF queries with rdma-based distributed graph exploration. In *12th USENIX Symposium on Operating Systems Design and Implementation (OSDI 16)* (Savannah, GA, Nov. 2016), USENIX Association, pp. 317–332.

[41] SHULL, T., HUANG, J., AND TORRELLAS, J. Autopersist: An easy-to-use Java NVM framework based on reachability. In *Proceedings of the 40th ACM SIGPLAN Conference on Programming Language Design and Implementation* (New York, NY, USA, 2019), PLDI 2019, Association for Computing Machinery, p. 316–332.

[42] SMOLYAR, I., MARKUZE, A., PISMENNY, B., ERAN, H., ZELLWEGER, G., BOLEN, A., LISS, L., MORRISON, A., AND TSAFRIR, D. Ioctopus: Outsmarting nonuniform dma. In *Proceedings of the Twenty-Fifth International Conference on Architectural Support for Programming Languages and Operating Systems* (New York, NY, USA, 2020), ASPLOS '20, Association for Computing Machinery, p. 101–115.

[43] TALEB, Y., STUTSMAN, R., ANTONIU, G., AND CORTES, T. Tailwind: Fast and atomic rdma-based replication. In *2018 USENIX Annual Technical Conference (USENIX ATC 18)* (Boston, MA, July 2018), USENIX Association, pp. 851–863.

[44] TALPEY, T., AND KAMER, G. High performance file serving with SMB3 and RDMA via SMB direct. In *Storage Developers Conference* (2012).

[45] THE H-STORE TEAM. SmallBank Benchmark. http://hstore.cs.brown.edu/documentation/deployment/benchmarks/smallbank/.

[46] THE TRANSACTION PROCESSING COUNCIL. TPC-C Benchmark V5.11. http://www.tpc.org/tpcc/.

[47] TSAI, S.-Y., AND ZHANG, Y. LITE kernel RDMA support for datacenter applications. In *Proceedings of the 26th Symposium on Operating Systems Principles* (New York, NY, USA, 2017), SOSP '17, Association for Computing Machinery, p. 306–324.

[48] VENKATARAMAN, S., TOLIA, N., RANGANATHAN, P., AND CAMPBELL, R. H. Consistent and durable data structures for non-volatile byte-addressable memory. In *Proceedings of the 9th USENIX Conference on File and Stroage Technologies* (USA, 2011), FAST'11, USENIX Association, p. 5.

[49] VOLOS, H., MAGALHAES, G., CHERKASOVA, L., AND LI, J. Quartz: A lightweight performance emulator for persistent memory software. In *Proceedings of the 16th Annual Middleware Conference* (New York, NY, USA, 2015), Middleware '15, Association for Computing Machinery, p. 37–49.

[50] VOLOS, H., TACK, A. J., AND SWIFT, M. M. Mnemosyne: Lightweight persistent memory. In *Proceedings of the Sixteenth International Conference on Architectural Support for Programming Languages and Operating Systems* (New York, NY, USA, 2011), ASPLOS XVI, Association for Computing Machinery, p. 91–104.

[51] WANG, S., LOU, C., CHEN, R., AND CHEN, H. Fast and concurrent RDF queries using rdma-assisted GPU graph exploration. In *2018 USENIX Annual Technical Conference (USENIX ATC 18)* (Boston, MA, July 2018), USENIX Association, pp. 651–664.

[52] WEI, X., CHEN, R., AND CHEN, H. Fast RDMA-based Ordered Key-Value Store using Remote Learned Cache. In *14th USENIX Symposium on Operating Systems Design and Implementation (OSDI 20)* (Nov. 2020), USENIX Association, pp. 117–135.

[53] WEI, X., DONG, Z., CHEN, R., AND CHEN, H. Deconstructing RDMA-enabled distributed transactions: Hybrid is better! In *13th USENIX Symposium on Operating Systems Design and Implementation (OSDI 18)* (Carlsbad, CA, Oct. 2018), USENIX Association, pp. 233–251.

[54] WEI, X., SHI, J., CHEN, Y., CHEN, R., AND CHEN, H. Fast in-memory transaction processing using RDMA and HTM. In *Proceedings of the 25th Symposium on Operating Systems Principles* (New York, NY, USA, 2015), SOSP '15, Association for Computing Machinery, p. 87–104.

[55] WU, M., ZHAO, Z., LI, H., LI, H., CHEN, H., ZANG, B., AND GUAN, H. Espresso: Brewing java for more non-volatility with non-volatile memory. In *Proceedings of the Twenty-Third International Conference on Architectural Support for Programming Languages and Operating Systems* (New York, NY, USA, 2018), ASPLOS '18, Association for Computing Machinery, p. 70–83.

[56] XIE, X., WEI, X., CHEN, R., AND CHEN, H. Pragh: Locality-preserving graph traversal with split live migration. In *2019 USENIX Annual Technical Conference (USENIX ATC 19)* (Renton, WA, July 2019), USENIX Association, pp. 723–738.

[57] XU, J., AND SWANSON, S. NOVA: A log-structured file system for hybrid volatile/non-volatile main memories. In *14th USENIX Conference on File and Storage Technologies (FAST 16)* (Santa Clara, CA, Feb. 2016), USENIX Association, pp. 323–338.

[58] YANG, J., IZRAELEVITZ, J., AND SWANSON, S. Orion: A distributed file system for non-volatile main memory and RDMA-Capable networks. In *17th USENIX Conference on File and Storage Technologies (FAST 19)* (Boston, MA, Feb. 2019), USENIX Association, pp. 221–234.

[59] YANG, J., KIM, J., HOSEINZADEH, M., IZRAELEVITZ, J., AND SWANSON, S. An empirical guide to the behavior and use of scalable persistent memory. In *18th USENIX Conference on File and Storage Technologies (FAST 20)* (Santa Clara, CA, Feb. 2020), USENIX Association, pp. 169–182.

[60] ZHANG, Y., CHEN, R., AND CHEN, H. Sub-millisecond stateful stream querying over fast-evolving linked data. In *Proceedings of the 26th Symposium on Operating Systems Principles* (2017), SOSP '17, p. 614–630.

[61] ZHANG, Y., YANG, J., MEMARIPOUR, A., AND SWANSON, S. Mojim: A reliable and highly-available non-volatile memory system. In *Proceedings of the Twentieth International Conference on Architectural Support for Programming Languages and Operating Systems* (New York, NY, USA, 2015), ASPLOS '15, Association for Computing Machinery, p. 3–18.

[62] ZIEGLER, T., TUMKUR VANI, S., BINNIG, C., FONSECA, R., AND KRASKA, T. Designing distributed tree-based index structures for fast rdma-capable networks. In *Proceedings of the 2019 International Conference on Management of Data* (New York, NY, USA, 2019), SIGMOD '19, Association for Computing Machinery, p. 741–758.

[63] ZUO, P., HUA, Y., AND WU, J. Write-optimized and high-performance hashing index scheme for persistent memory. In *13th USENIX Symposium on Operating Systems Design and Implementation (OSDI 18)* (Carlsbad, CA, Oct. 2018), USENIX Association, pp. 461–476.