Exploring Efficient Architectures on Remote In-Memory NVM over RDMA

QINGFENG ZHUGE, HAO ZHANG, EDWIN HSING-MEAN SHA, and RUI XU,
School of Computer Science and Technology, East China Normal University, China
JUN LIU and SHENGYU ZHANG, Levono Ltd., China

Efficiently accessing remote file data remains a challenging problem for data processing systems. Development of technologies in non-volatile dual in-line memory modules (NVDIMMs), in-memory file systems, and RDMA networks provide new opportunities towards solving the problem of remote data access. A general understanding about NVDIMMs, such as Intel Optane DC Persistent Memory (DCPM), is that they expand main memory capacity with a cost of multiple times lower performance than DRAM. With an in-depth exploration presented in this paper, however, we show an interesting finding that the potential of NVDIMMs for high-performance, remote in-memory accesses can be revealed through careful design. We explore multiple architectural structures for accessing remote NVDIMMs in a real system using Optane DCPM, and compare the performance of various structures. Experiments are conducted to show significant performance gaps among different ways of using NVDIMMs as memory address space accessible through RDMA interface. Furthermore, we design and implement a prototype of user-level, in-memory file system, RIMFS, in the device DAX mode on Optane DCPM. By comparing against the DAX-supported Linux file system, Ext4-DAX, we show that the performance of remote reads on RIMFS over RDMA is 11.44 × higher than that on a remote Ext4-DAX on average. The experimental results also show that the performance of remote accesses on RIMFS is maintained on a heavily loaded data server with CPU utilization as high as 90%, while the performance of remote reads on Ext4-DAX is significantly reduced by 49.3%, and the performance of local reads on Ext4-DAX is even more significantly reduced by 90.1%. The performance comparisons of writes exhibit the same trends.

CCS Concepts: • **Computer systems organization → Architectures;**

Additional Key Words and Phrases: Non-volatile memory, remote access, RDMA

ACM Reference format:
Qingfeng Zhuge, Hao Zhang, Edwin Hsing-Mean Sha, Rui Xu, Jun Liu, and Shengyu Zhang. 2021. Exploring Efficient Architectures on Remote In-Memory NVM over RDMA. *ACM Trans. Embedd. Comput. Syst.* 20, 5s, Article 73 (September 2021), 20 pages.
https://doi.org/10.1145/3477004

Authos' addresses: Q. Zhuge, H. Zhang, E. H.-M. Sha (corresponding author), and R. Xu, School of Computer Science and Technology, East China Normal University, China; emails: qfzhuge@cs.ecnu.edu.cn, 51194506046@stu.ecnu.edu.cn, edwinsha@cs.ecnu.edu.cn, ruixu@stu.ecnu.edu.cn; J. Liu and S. Zhang, Levono Ltd., China; emails: {liujun8, zhangsy19}@lenovo.com.

1539-9087/2021/09-ART73 $15.00
https://doi.org/10.1145/3477004

1 INTRODUCTION

Accessing remote data through fast network is a critical and challenging issue for systems with high-performance data storage and processing requirements. The performance barriers exist in both the networks and the data storage systems in some widely used distributed file systems, such as HDFS [1, 2] and Ceph [3]. For example, the latency of advanced SSDs is still about 5 × exceeds the network latency, while the fast DRAM memory is difficult to be used as storage due to its volatility and small capacity. The emergence of Non-volatile memory (NVM) technologies with nanosecond latency, such as Racetrack Memory [4, 5], Phase Change Memory (PCM) [6], Memristor [7] and 3D XPoint [8], close the performance gap of data storage systems with features of data durability and high capacity [9, 10]. New in-memory file systems, including SIMFS [11], PMFS [12], SCMFS [13] and Strata [14], are designed to exploit the performance advantages of the byte-addressable, persistent data storage that made possible by NVMs in main memory.

The low-latency data storage systems need to work together with high-performance networks, such as Remote Direct Memory Access (RDMA) technology [15], to provide high capacity of data transmission. Theoretically speaking, a remote data access request can be directly performed as a data access in the main memory on server through RDMA without involving the CPU. However, the integration of RDMA networks and file systems on NVM are not fully studied. It is necessary to understand how the system performance gain can be achieved with architectures that employ both RDMA networks and NVMs. In literature, a few of research works have been conducted to explore the opportunities of remote file systems on architectures combining RDMA networks and NVMs, such as Octopus [16], Orion [17], and NVFS [18]. However, these works use DRAM memory to simulate NVMs in main memory, and indicate the results can be extended to real NVM main memory. As a result, the characteristics of real NVM modules connected to memory bus via DIMMs, aka non-volatile dual in-line memory modules (NVDIMMs), can not be considered in their design. At present, few work has explored the combination of NVDIMMs and RDMA networks. Actually, it is of vital interest to conduct further studies on real NVDIMM devices to explore the architectural features, understand their effects on system performance, and then guide the design of in-memory file systems that can fully exploit the advantages of NVDIMMs and RDMA networks.

In this paper, we explore various architectural structures to achieve high-performance, remote data access with Intel Optane DC Persistent Memory (Optane DCPM), the first commercial NVDIMM product at present, and RDMA network. NVM devices can be accessed by RDMA only if they can be managed as main memory in the virtual memory address space, which are similar to Optane DCPM. we will explore a general solution for NVM over RDMA by studying Optane DCPM. We conduct extensive experiments to show the challenges and opportunities of building high-performance file system with NVDIMMs that can be efficiently accessed over RDMA. The contributions of this paper are as follows:

(1) We explore different data access modes on Optane DCPM, in particular, FSDAX and DEVDAX namespaces, to access remote memory over RDMA. We find that significant overhead incurred under FSDAX mode that is caused by the lack of page table for data access in memory address space.
(2) We explore various architectural structures of accessing Optane DCPM in memory address space over RDMA. We show that the performance gap between DRAM and Optane DCPM is significantly reduced for remote data accesses over RDMA. And for Optane DCPM, the performance of remote data accesses over RDMA is even greater than that of local data accesses on large IO size.
(3) Based on the exploration of architectural features for remote data accesses on NVDIMM over RDMA, we design and implement a user-level, in-memory file system, named as Remote

in-Memory File System (RIMFS), to achieve high-performance, remote data access. Experimental results on the prototype of RIMFS show that the throughput of remote reads on RIMFS is 11.4 × higher than that of Ext4-DAX over RDMA on average. The throughput of remote writes on RIMFS is 12.8 × higher than that of Ext4-DAX over RDMA.

(4) We conduct experiments to evaluate performance of file systems when data server is heavily loaded with various CPU utility. The experimental results show that the performance of RIMFS is hardly reduced when the CPU utility is as high as 90% on data server, while the performance of remote reads on Ext4-DAX is significantly reduced by 49.3%, and the performance of local reads on Ext4-DAX is even more significantly reduced by 90.1%.

This paper proceeds as follows. Section 2 provides the background of NVM, RDMA, and related works. The motivation of our exploration in this paper is described in Section 3. Section 4 presents configuration details on the evaluation environment and some symbols used in this paper. Section 5 focuses on exploring the architecture of NVM over RDMA. Section 6 presents a file system prototype based on the architecture of DEVDAX over RDMA, and some evaluations of this file system. And Section 7 concludes this paper.

2 BACKGROUND AND RELATED WORK

2.1 Intel Optane DC Persistent Memory

In recent years, many NVM technologies are widely studied and used, such as Racetrack Memory [4, 5], PCM [6], Memristor [7] and 3D XPoint [8]. NVM provides data persistence while achieving latency and bandwidth close to DRAM. What's more, NVM has higher density and lower price than DRAM.

The Intel Optane DC Persistent Memory (Optane DCPM) [19], shipped in 2019, is the only NVDIMM device currently available on the market. The capacity of Optane DCPM is larger than that of traditional DRAM. Concretely, the capacity of Optane DCPM can achieve up to 512 GB, which is larger than the commonly used DRAM DIMM, although DRAM has 128 GB DIMM. Meanwhile, Optane DCPMs can be accessed by CPU via byte-addressable instructions, such as load and store, by attaching directly to the memory bus, and the latency is only a few hundred nanoseconds. Although Optane DCPM has advantages mentioned above, the latency and bandwidth are still worse than that of DRAM. Thus, it is always used to expand the volume of main memory in addition to DRAM. One popular configuration of the ratio between DRAM and NVM recommended by Intel is the 1:4 with 128 GB Optane DCPM per 32 GB DRAM. Each Optane DCPM is internally divided into blocks of 256 bytes, and the controller inside Optane DCPM with access granularity of 64 bytes would translate commands into 256 bytes' reads and writes to these blocks. Additionally, the controller implements wear-leveling and faulted block management. Because Optane DCPMs are connected to the memory bus, they can be accessed directly from other devices that support data memory access (DMA).

Optane DCPM can be configured in two modes: memory mode and app-direct mode. In memory mode, Optane DCPM is used as a DRAM and does not provide data persistence, while DRAM is used as a cache for Optane DCPM. In app-direct mode, user processes can access Optane DCPM directly; Optane DCPM can provide data persistence; and Optane DCPM can work in four namespaces: fsdax, devdax, sector, and raw.

(1) **fsdax:** The Optane DCPM in fsdax namespace (FSDAX) is a block device that supports mounting Linux file system with DAX feature. DAX removes the page cache from the I/O path and allows mmap to establish a direct mapping to persistent memory media.

(2) **devdax:** The Optane DCPM in devdax namespace (DEVDAX) is a character device that supports DAX mapping similar to the DAX-supported file system. But file system can not mount

on DEVDAX, memory addresses in DEVDAX are allocated to user processes by mmap instead. Persistent memory in DEVDAX can be registered for RDMA.

(3) **sector:** The sector namespace can mount legacy file systems or be used as small boot volumes.

(4) **raw:** The raw namespace is just a memory disk that does not support DAX.

2.2 RDMA

Remote Direct Memory Access (RDMA) [15] technology has attracted much attention because of its low latency and high bandwidth. It is a host-offload, host-bypass and zero-copy technology that allows to directly access remote memory in user-level without remote CPU participation, in which the Ethernet card with a RDMA engine (RNIC) is responsible for managing the reliable connection between the source and the target. This technology greatly reduces the number of data copies and software overheads during communication, thus increasing the rate of communication. For example, a packet that transmits 512 bytes between two nodes in a data center has a latency of about 4 *us*, while the transmission latency of a traditional Ethernet is 114 *us* in the same case. At the same time, the newly listed ConnectX-6 [20] network interface controllers and ConnectX-5 network interface controllers supporting RDMA have been able to support the bandwidth of 200 *Gbps* and 100 *Gbps*, respectively. RDMA provides its special usage mechanism, the most important of which includes *Memory Registration* and *Verbs*, as follows:

Memory Registration: *Memory Registration* is a measure of memory protection in RDMA. The registration process pins the memory pages to prevent the pages from being swapped out and to keep mapping between physical addresses and virtual addresses, and copies the address mapping table to the RNIC. The registered memory area is called *Memory Region*. *Memory Registration* is necessary, because the RDMA hardware has special requirements for memory used for data transmission: (1) The access permissions for the memory need to be set in the *Memory Region*, including local write, remote read, remote write, atomic and bind. (2) The operating system cannot page out the memory where the data is located - the mapping between physical and virtual addresses must be fixed.

Verbs: The application interact with the RNIC through *RDMA Verbs* API. Send/Recv *verbs* are often called two-sides operation, because each Send operation requires a corresponding Recv operation match on the remote machine and remote CPU participation. Read/Write *verbs* are called one-side operation, because the local machine can access the remote memory directly through the remote RNIC, without participation of remote CPU.

2.3 Persistent RDMA Write

When writing data to a remote NVM by *RDMA write*, the data may be not actually stored in the NVM. [21] The data may be cached at any buffer on the path, such as LLC, PCIe buffer, and RNIC buffer et al. But the data in the cache is volatile, and will lose upon system crash. To persist data, the influence of caching should be avoided. Fortunately, this problem would be solved by combining the following two techniques.

Turn off DDIO: Some Intel CPUs have Data Direct I/O (DDIO) technology [22]. When DDIO is on, data written by RNIC to NVM will be cached in CPU caches, and will be volatile. So it is necessary to prevent data from being written to CPU caches by turning off DDIO.

Flush buffer: After issuing a series of *RDMA write* operations, the client needs to initiate an *RDMA Read* operation with any address and content on the same RDMA connection to complete data persistence. This method ensures that the data will enter the destination, such as NVM, after it arrives the remote machine. This method [21] is guaranteed by PCIe based I/O transactions, and

the Intel Integrated I/O in their hardware will flush all previous *PCIe Writes* to the destination before executing the *PCIe Read* request.

2.4 Related Work

Previously, many works [23–26] show that RDMA can improve system performance. FaRM [23] is a memory distributed computing platform based on RDMA; Jose et al. [24] design a new Memcached, a key-value distributed memory object caching system using RDMA networks; and they [25] use RC and UD of RDMA to increase performance of Memcached; i10 [26] is a remote storage stack to make the performance of TCP similar to the performance of RDMA. Although, these works explore the capacity of the architecture of RDMA, and compare it with the state-of-the-art architecture over TCP. They do not explore the performance of of RDMA under different architectures, such as different memory technologies over RDMA.

There are previous works [9, 10, 12, 14] to propose data storage system on NVMs that is attached to memory bus. Yang et al. [9] evaluate the performance of real NVM in some situations including RockDB and PMemKV; Mason et al. [10] use Polybench Test to compare the performance of NVM with that of DRAM; SIMFS [11] reconstructs the file metadata by using memory mapping hardware to make full use of the byte-addressability of NVM; PMFS [12] is a lightweight POSIX file system to enable the persistent memory accessed by application directly; SCMFS [13] leverages the virtual memory management in the operating system to keep the data space in NVM contiguous for each file; Strata [14] is a file system for hybrid storage, including NVM, SSD, and HDD. Works mentioned above are only exploring the performance of NVM. But there are a little works for RDMA directly accessing NVM. Furthermore, the characteristics of real NVM devices are still not clear now. And Optane DCPM is the only commercial product of NVM. Thus, there is not much work that can clearly explore the application of NVM. Kalia et al. [27] study writing data to NVM with low latency at the small granularity and high bandwidth at large granularity by RPC using RDMA. However, they have no regard for the specific modes and different configurations under RDMA accessing directly.

To well exploit the combination of NVM and RDMA, amounts of works [16, 28–31] have explored the file system designs for this architecture. Crail [28, 29] is a distributed file system based on DaRPC [30] which is an RPC library optimized by RDMA and is designed for NVMe over RDMA. NVFS [18] is a file system optimized by RDMA and NVM based on HDFS. Orion [17] is a log-structured distributed file system for NVM and RDMA-capable networks. Octopus [16] is a decentralized distributed file system for NVM and RDMA. Mojim [31] utilizes NVM to provide data replica stored in data center, and uses RDMA to avoid the long latency of data replication. But all of them use DRAM to simulate NVM for persistent storage, or only use DRAM for volatile storage. And to make NVM similar to DRAM under RDMA access, some memory architectures related issues need to be resolved. Therefore, this paper will explore a general solution for NVM over RDMA by studying Optane DCPM.

Most of the existing works use DRAM to simulate NVM, but they cannot accurately show the characteristics of real NVM. Therefore, there are still many points to be explored for the real NVM. Fortunately, we observe that the performance of the architecture that NVM is directly attached to the memory bus and is accessed by RDMA is inspiring. Furthermore, to show the data processing capabilities of this architecture, such as low overhead, low latency, high throughput et al. a file system prototype for this architecture is designed.

3 MOTIVATION

DRAM simulated NVM is limited. In this experiment, the performance of copying data with a granularity of 8 bytes are evaluated with various copy directions to show the performance of data

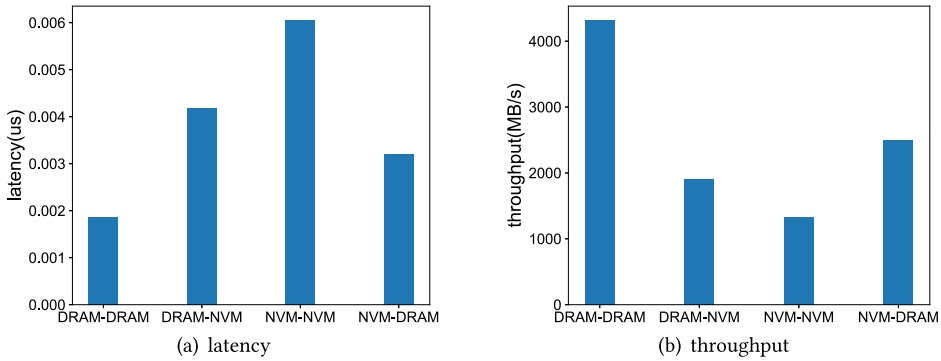


Fig. 1. Comparison of the latency and throughput of memory copy with 8-byte under different memory copy paths (DRAM-DRAM, DRAM-NVM, NVM-NVM, NVM-DRAM).

movements on DRAM and NVM. For example, experimental data of copying data from DRAM to NVM is represented by the label DRAM-NVM. Similarly, the other data copy directions are labeled as DRAM-NVM, NVM-DRAM, and NVM-NVM.

As shown in the Figure 1, the results show that the latency of writing data from DRAM to NVM, shown as DRAM-NVM, is 30% higher than that of reading data from NVM to DRAM, shown as NVM-DRAM. And compared with DRAM-DRAM, the latency of NVM-NVM, DRAM-NVM, and NVM-DRAM are 226.2%, 125.9%, and 73.2% higher, respectively. At the same time, recent studies [17, 32] have given the corresponding performance results: (1) **latency** - the write latency of NVM is 250 ns, the read latency of NVM is 150 ns, while the read and write latency of DRAM are 10 $\sim$ 20 ns. (2) **bandwidth** - the read and write bandwidths of NVM are 8 GB/s and 2 GB/s, respectively. The read and write bandwidths of DRAM are 60 GB/s and 30 GB/s, respectively. Therefore, both the evaluation results and other works show that DRAM is different from NVM in performance. Besides, NVM has the feature of non-volatility and has some mechanisms to persist data. But DRAM is volatile, it is impossible to be the same as NVM, when using DRAM to simulate NVM. Therefore, it is necessary to explore the characteristics on real NVM devices.

RDMA is still better than TCP under the memory architecture with NVM. Although many studies have shown that RDMA performs better than TCP [26], the research is few for exploring the performance of NVM over RDMA. The following experiment shows that the performance of RDMA transmission is better than that of TCP transmission. In this experiment, the remote procedure call (RPC) is used for TCP to read data from remote NVM, which is described as TCP_r; and the *RDMA read* is used for RDMA to read remote data, which is described as $RDMA_r$. As shown in Table 1, the latency of of TCP_r is orders of magnitude times the latency of $RDMA_r$. Specifically, the transmission overhead of TCP_r is much higher than that of $RDMA_r$ on small IO size, and the latency of TCP_r is 44.92 $\times$ higher than $RDMA_r$ on 512 bytes IO size.

Remote is better than local. Generally, accessing memory locally is considered faster than accessing memory through the network. However, in the experiment, we find that when the IO size is 2 MB, accessing NVM over RDMA has lower latency than accessing NVM locally. For read, the latency of accessing NVM over RDMA is 61.3% lower than the latency of accessing NVM locally; and for write, this ratio is 37.9%.

According to above experiments, we find that: (1) The read/write characteristics of NVM are completely different from DRAM. (2) When accessing NVM, RDMA can achieve performance far exceeding traditional TCP/IP. (3) The performance of NVM under RDMA access may exceed the performance of local accessing NVM. Based on learning the characteristics of NVM and RDMA comprehensively, the **objective** of this paper is to explore an architecture to maximize the

Table 1. Comparison of the Read Latency (us) from Remote NVM for TCP and RDMA

	TCP_r	$RDMA_r$
512 B	112.6	2.5
4 KB	211.9	3.9
128 KB	1367.3	18.7
2 MB	19634.8	228.5

The latency of TCP is orders of magnitude times the latency of RDMA.

Table 2. Configuration Details of Server and Client

	Server	Client
CPU	Intel(R) Xeon(R) Gold 5218 CPU @ 2.30GHz, 16 cores	Intel(R) Core(TM) i5-9500 CPU @ 3.00GHz, 6 cores
DRAM	192 GB DDR4 DIMM (32 GB/DIMM)	16 GB DDR4 DIMM (16 GB/DIMM)
NVM	512 GB Intel Optane DIMM (128 GB/DIMM)	-
OS	Ubuntu 20.04 with linux kernel version 5.8.0	
NIC	Mellanox Technologies MT27800 Family [ConnectX-5] (100 $Gbps$)	

performance of NVM over RDMA. And based on the architecture, a file system prototype RIMFS is designed and implemented.

4 EVALUATION SETUP

The experiments in this paper are carried out on a cluster with 1 server (ThinkSystem SR650) and 8 clients, all of which are connected to each other by a switch Mellanox SB 7890. The non-blocking bandwidth of switch is 7 Tb/s. The port-to-port latency is 90 ns, and each port provides the full bidirectional bandwidth of the highest 100 $Gbps$. Table 2 reports the detailed configuration of the relevant server and each client.

As shown in Table 3, in order to facilitate the description of the subsequent experiments, some symbols will be used to replace the description of the experimental subjects in this paper.

5 EXPERIMENT AND DISCUSSION

In this section, the characteristics of real in-memory NVM device and several architectures under NVM over RDMA are explored and evaluated to illustrate the advantage when NVM is accessed by RDMA. And the symbols used in follows are shown in Table. 3. The data access methods of FSDAX and DEVDAX are different: for FSDAX, data access will pass through the mounted file system; for DEVDAX, data access will pass through the page table. Therefore, these two different ways of data access will have different performance. According to the experimental results, the performance of FSDAX is worse than that of DEVDAX, and NVM can play a better performance close to DRAM, when accessed by RDMA. And 5 critical observations are obtained from the experiments to prove the conclusions mentioned above and promote our design of in-memory file system based on NVM over RDMA.

5.1 Fsdax vs. Devdax

In this section, the characteristics and performances of $FSDAX_{RDMA}$ and $DEVDAX_{RDMA}$ are explored and evaluated. Then, there are two observations including the performances under two different architectures as following.

Table 3. Symbol and Description

Symbol	Description	Symbol	Description
FSDAX	The NVM in fsdax namespace	DEVDAX	The NVM in devdax namespace
$DEVDAX_{RDMA}$	Access the data in DEVDAX by RDMA	$DEVDAX_{LOC}$	Local access the data in DEVDAX
$DRAM_{RDMA}$	Access the data in DRAM by RDMA	$DRAM_{LOC}$	Local access the data in DRAM
$FSDAX_{RDMA}$	Access the data in FSDAX by RDMA		
ODP	On-Demand-Paging is a technique to ease the RDMA memory registration (Section 5.1 for detail).		
IMP_{DEVDAX}	The latency improvement of $DEVDAX_{RDMA}$ comparing with $DEVDAX_{LOC}$		
IMP_{DRAM}	The latency improvement of $DRAM_{RDMA}$ comparing with $DRAM_{LOC}$		

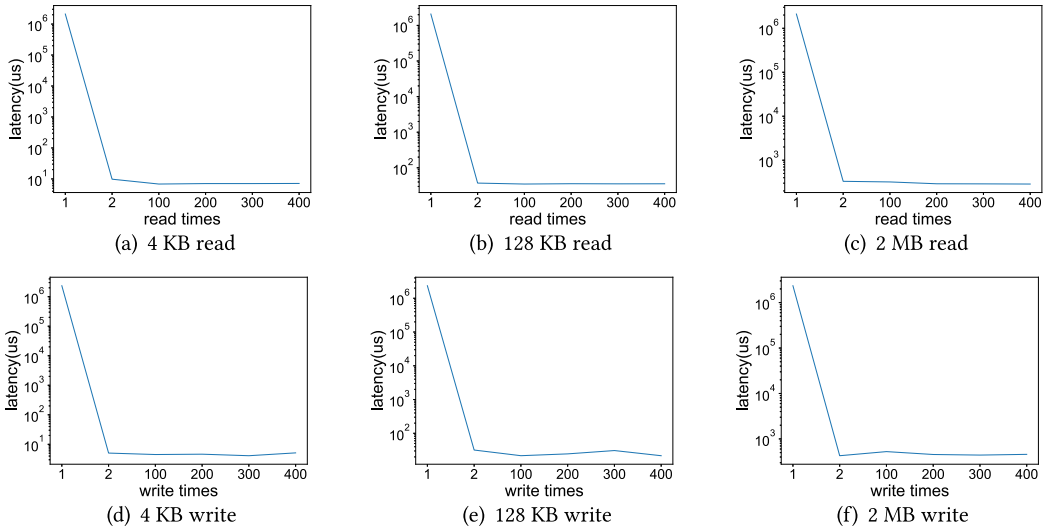


Fig. 2. Comparison of the read/write latency for FSDAX which is accessed by RDMA. The results represent the latency for reading the same data at the nth time where n is from 1 to 400. The latency of the first access far exceeds the latency of the later access.

Observation 1: the access latency to un-accessed regions in remote FSDAX is particularly high, and this latency is almost independent of the size of region to be accessed. Figure 2 shows the changes in the read/write latency for FSDAX when reading the same data repeatedly at different times, and the IO sizes are set to 4 KB, 128 KB, and 2MB, respectively. As shown in Figure 2, for every IO size, as the first time read/write, the access latency are extremely high. For *RDMA read* on different IO sizes, the latency is about 2.14 *s*; and for *RDMA write* on different IO sizes, the latency is about 2.36 *s*. Furthermore, all of the read/write latency are sharply decline and trend to be stable.

One main reason of the high access latency is: a page fault exception occurs when RDMA accessing an un-accessed region in the FSDAX. There should be a DAX-supported file system mount on the FSDAX, in which the I/O path will not pass through page tables. Therefore, the operating system (OS) does not establish page tables for the region in FSDAX. However, page tables is necessary on RDMA access, and when the *Memory Region* is constructed, the related page tables need to be copied into the RNIC. Therefore, to support the function of access over RDMA in FSDAX, ConnectX-5 and newer versions RNICs provide a mechanism called on-demand-paging (ODP).

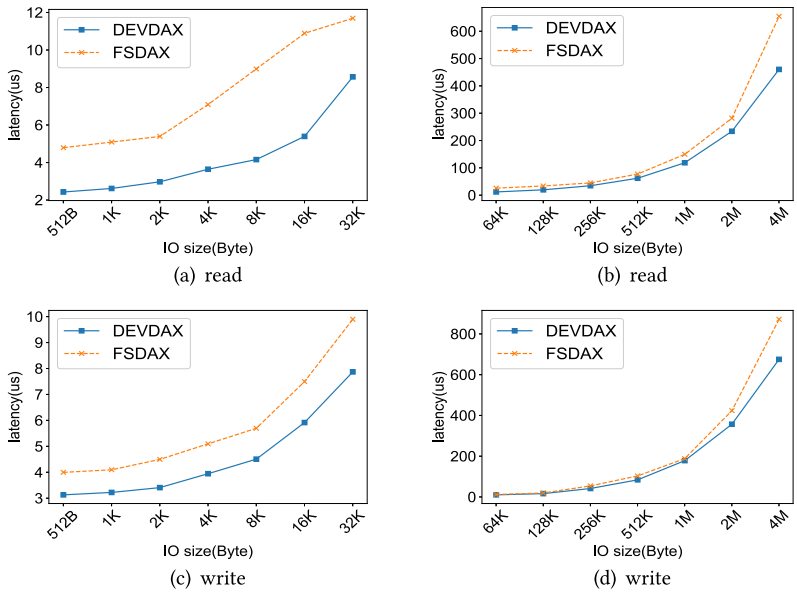


Fig. 3. Comparison of the read/write latency for DEVDAX and FSDAX under RDMA access. The latency of DEVDAX is lower than that of FSDAX whatever the IO size is.

And when registering the *Memory Region* for the region in FSDAX, the ODP parameter is needed to be set, but the RNIC will not cache the page table of the *Memory Region* which has the ODP parameter. So when the region, whose page tables are not established or cached by RNIC, is accessed over RDMA, the RNIC will ask the OS for page translations [33]. Then, the OS throws page fault exceptions to build the page table for that region. This process requires the participation of the page fault mechanism of the OS, so that the process will be time-consuming. This overhead is several orders of magnitude higher than that of RDMA data transmission, so the latency of RDMA transmission is hided, and the access latency seems to be same whatever the IO size is.

After the page tables are established during the processing for page fault, they will be copied into RNIC. Therefore, the page fault will not occur again when the same region is accessed. As shown in Figure 2, the latency is decreased by several orders of magnitude for second access, and keeps stable in following accesses.

Observation 2: DEVDAX has lower latency than FSDAX under RDMA access. To avoid the influence of high cost caused by page fault and verify the latency of accessing the region where no page fault exception would happen, in FSDAX over RDMA, the following experiment is constructed. In this experiment, $FSDAX_{RDMA}$ is carried out in the region where the page table has been established, $DEVDAX_{RDMA}$ and $FSDAX_{RDMA}$ are evaluated, respectively. As shown in the Figure 3, for the read latency, DEVDAX is 37.5% lower than FSDAX on average; specifically, the latency of DEVDAX is 29.4% lower than FSADX on 4 MB IO size. For the write latency, DEVDAX is 19.1% lower than FSDAX on average; specifically, DEVDAX is 22.5% lower than FSADX on 4 MB IO size. Overall, the latency of DEVDAX is always lower than that of FSDAX under RDMA access.

5.1.1 Discussion for FSDAX. FSDAX is convenient to maintain data and also programmer-friendly. However, the latency under RDMA access is too high to be ignored and the reasons behind this are two characters of FSDAX as follows:

Block device: because FSDAX belongs to a block device, the data I/O path will pass through the file system mounted on FSDAX, and data are also stored as files. This makes it convenient for the

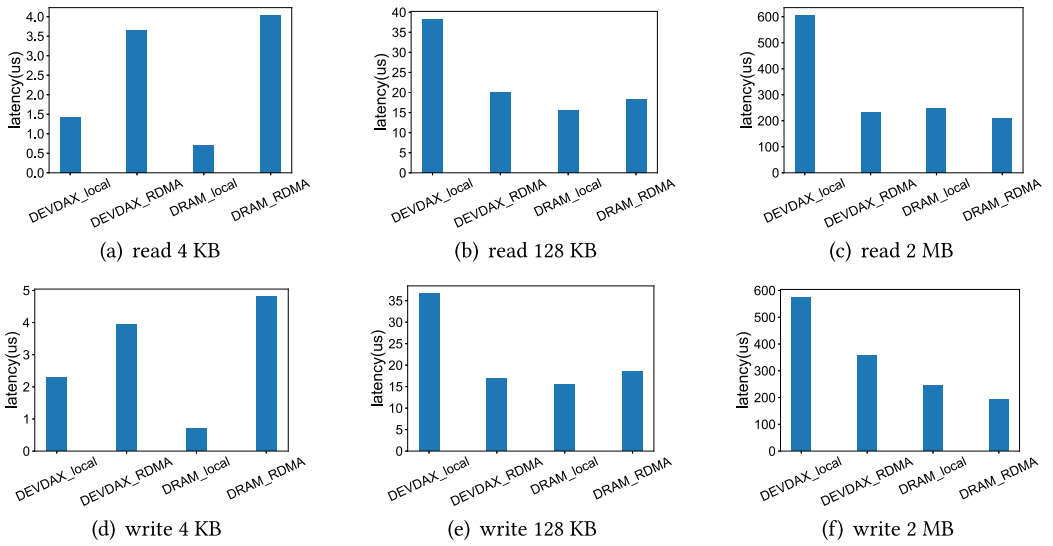


Fig. 4. Comparison of the read/write latency under different architectures (DEVDAX_local, DEVDAX_RDMA, DRAM_local, DRAM_RDMA).

server to maintain data because the organization of metadata is already done by the file system. Also, programmers can use the POSIX interface to access FSDAX locally without modifying any code. But, this feature is extremely unfriendly to RDMA access. This is because (1) when the size of a file in FSDAX changes, the *Memory Region* for the file needs to be rebuilt. And the cost of building *Memory Region* is huge. According to the experiment, when 4 KB data is accessed by RDMA, the cost of building *Memory Region* accounts for more than 90% of the total latency. (2) the file system will bring more software overhead to access, such as the copy of metadata from the file system to the user process. Undoubtedly, both of them bring troubles to access FSDAX over RDMA. To avoid the cost of rebuilding *Memory Region*, a solution of pre-allocating and registering the *Memory Region* is considered as our future work.

ODP: the page table will not be built for the region in FSDAX with ODP parameter on *Memory Registration*. So page fault exception will be thrown by the OS when accessing the region without page table over RDMA. Then, the OS needs to allocate page tables for this region, resulting in huge overheads (as can be seen from the previous experiment in Section 5.1).

5.1.2 Discussion for DEVDAX. DEVDAX is more similar with DRAM relative to FSDAX, thus it can exploit the high performance of RDMA more comprehensively. This reason is that DEVDAX is a character device, and the data I/O path in DEVDAX will pass through page tables like DRAM. So the ODP flag does not need to be set, when registering *Memory Region* for the region in DEVDAX, and there is no overhead caused by page fault. Therefore, accessing DEVDAX over RDMA is efficient, and we recommend storing the persistent data in DEVDAX for RDMA access.

5.2 Improvement of DEVDAX by RDMA

Observation 3: the performance improvement of DEVDAX over RDMA to local DEVDAX is greater than that of DRAM over RDMA to local DRAM. In this experiment, the read/write latency of $DEVDAX_{LOC}$, $DEVDAX_{RDMA}$, $DRAM_{LOC}$ and $DRAM_{RDMA}$ is evaluated. To describe the improvement of performance, IMP_{DEVDAX} is used to express the latency improvement of $DEVDAX_{RDMA}$ comparing with $DEVDAX_{LOC}$, and the same as IMP_{DRAM}. As shown in Figure 4,

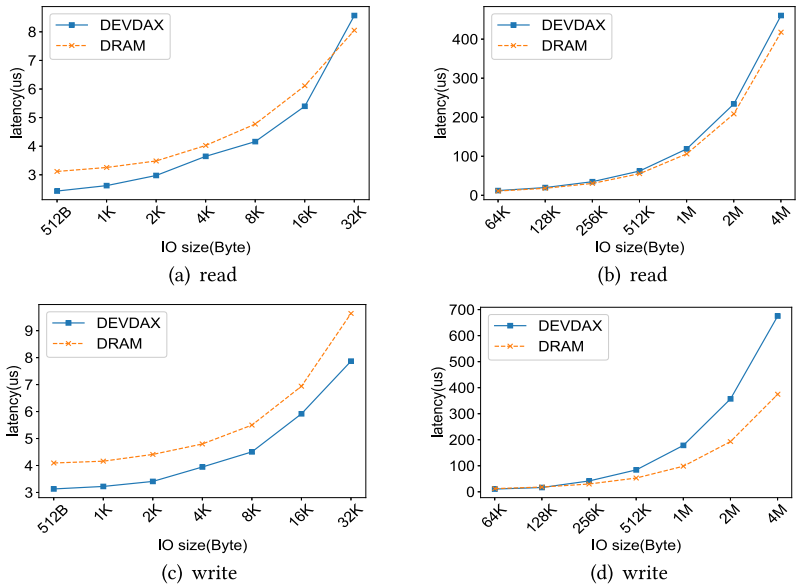


Fig. 5. Comparison of the read/write latency for DEVDAX and DRAM under RDMA access. The read/write latency of DRAM is higher than DEVDAX on small IO size. But on large IO size, the read latency of DEVDAX is close to that of DRAM, and the write latency of DEVDAX exceeds that of DRAM.

IMP_{DEVDAX} is higher than IMP_{DRAM}. For read/write, IMP_{DEVDAX} is 65.7% and 73.9% higher than IMP_{DRAM} on 128 KB IO size, respectively; and even when the IO size is 2 MB, IMP_{DEVDAX} is 46.1% and 16.3% higher than IMP_{DRAM}, respectively. But when the IO size is 4 KB, regardless DEVDAX or DRAM, the local access latency is lower than the RDMA access latency.

Observation 4: the latency of DEVDAX over RDMA is close to the latency of DRAM over RDMA. The latency of $DEVDAX_{RDMA}$ and $DRAM_{RDMA}$ is compared in the following experiment. As shown in Figure 5, for *RDMA read*, When the IO size is smaller than 32 KB, the latency of DEVDAX is 15% lower than that of DRAM on average; when the IO size is larger than 32 KB, the latency of DEVDAX is higher than that of DRAM, and on the 2 MB IO size, this gap is 12.1%. For *RDMA write*, when the IO size is smaller than 256 KB, the latency of DEVDAX is 17.9% lower than that of DRAM on average; when the IO size is larger than 256 KB, the latency of DEVDAX is higher than that of DRAM, and on the 2 MB IO size, this gap is 84.9%. Therefore, under RDMA access, DEVDAX can even achieve lower latency than DRAM on small IO size, and when the IO size exceeds a certain granularity, the latency of DEVDAX will be lower than that of DRAM.

Observation 5: accessing DEVDAX over RDMA can make DEVDAX perform better than local access in the case of larger IO size. To further explore the maximum performance of DEVDAX under RDMA access, the latency of $DEVDAX_{LOC}$ and $DEVDAX_{RDMA}$ are compared in the following experiment. As shown in the Figure 6, compared to the read latency of $DEVDAX_{LOC}$, (1) when the IO size is smaller than 32 KB, the read latency of $DEVDAX_{RDMA}$ is higher; specifically, when the IO size is 16 KB, the read latency of $DEVDAX_{RDMA}$ is 7.5% higher. (2) when the IO size is greater than 32 KB, the read latency of $DEVDAX_{RDMA}$ is lower; specifically, when the IO size is 4 MB, the read latency of $DEVDAX_{RDMA}$ is reduced by 61.1%. Compared to the write latency of $DEVDAX_{LOC}$, (1) when the IO size is smaller than 32 KB, the write latency of $DEVDAX_{RDMA}$ is higher; specifically, when the IO size is 16 KB, the write latency of $DEVDAX_{RDMA}$ is 12.6% higher. (2) when the IO size is greater than 32 KB, the write latency of $DEVDAX_{RDMA}$ is lower; specifically,

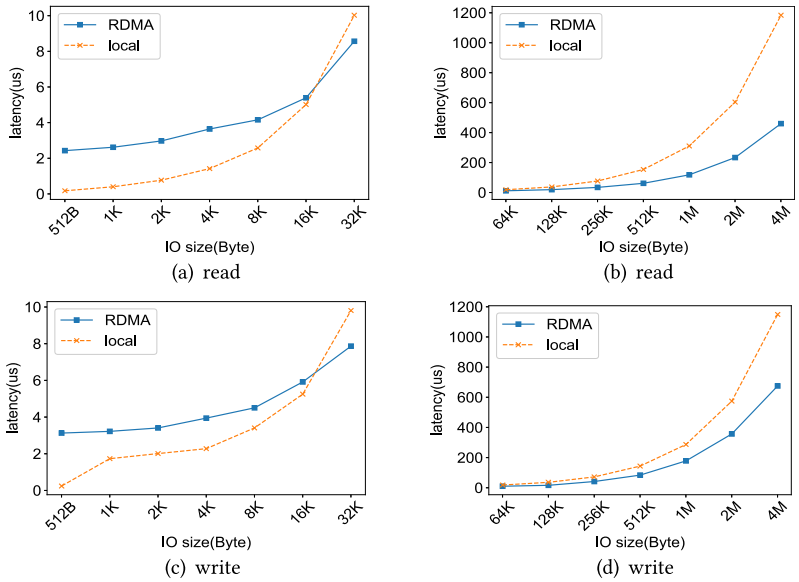


Fig. 6. Comparison of the read/write latency for DEVDAX over RDMA and local DEVDAX. The read/write latency of DEVDAX over RDMA is higher than local DEVDAX on small IO size. But on large IO size, the read/write latency of DEVDAX over RDMA is lower than that of local DEVDAX.

when the IO size is 4 MB, the write latency of $DEVDAX_{RDMA}$ is reduced by 41.3%. The **reasons** of the results mentioned above are:

(1) When the IO size is small, the main bottleneck of RDMA is network where data should be copied several times during transmission.
(2) As the increasing of IO size, the overhead of network is hided by the huge latency. And the advantage of high bandwidth of RDMA network can be better reflected.

To sum up, both the FSDAX and DEVDAX can be accessed by RDMA. But due to the block device nature of FSDAX, there are huge overheads when the region in FSDAX is accessed by RDMA, and the performance of RDMA and NVM can not be exploited. DEVDAX, as a character device with similar characteristics to DRAM, can achieve the following performance advantages: (1) DEVDAX achieves a performance similar to that of accessing DRAM under RDMA read. (2) When IO size is large, the latency of accessing DEVDAX is higher than that of DRAM under RDMA write. (3) when the IO size is large, the latency of accessing DEVDAX over RDMA is lower than that of local accessing DEVDAX. Therefore, DEVDAX is more suitable for RDMA access than FSDAX, and can play the best performance in case of large-granularity I/O requests. In addition, it is very necessary to design a low-overhead file system in user mode based on the architecture of DEVDAX over RDMA, due to following reasons: (1) existing file systems cannot mount on DEVDAX; (2) there is currently no file system that can manage data under the architecture of DEVDAX over RDMA with low overhead; (3) the zero-copy feature of RDMA accessing DEVDAX needs to be fully considered in the design of the file system. In the following section, we will present a low-latency and low-overhead in-memory file system prototype, to provide data management, RDMA access and other functions.

6 RIMFS

As analyzed in Section 5.1, DEVDAX is considered to be the best choice to design with RDMA. Unfortunately, there is no file system based on the architecture of DEVDAX over RDMA. Therefore,

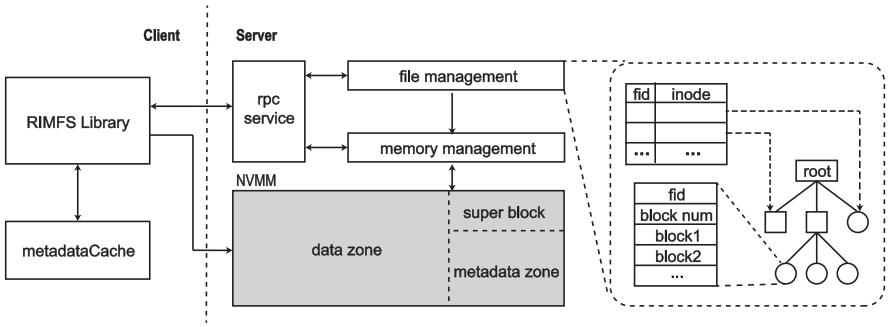


Fig. 7. RIMFS Overview.

this paper presents a file system prototype - Remote In-Memory File System (RIMFS). RIMFS is dedicated to the architecture of DEVDAX over RDMA which has low overhead, low latency, high bandwidth with low requirements for CPU capacity.

6.1 Overview

The overview of RIMFS is shown in Figure 7.

RIMFS consists of two parts: clients and a server. On the server-side, data blocks and metadata are stored and managed in the server's DEVDAX. To implement the data transmission between clients and a server over RDMA, RDMA Send/Recv verbs are used for communication, and RDMA Read/Write verbs are used for data access. To be detailed, RIMFS is divided into data management and data access modules described below.

Data management: RIMFS is a user-mode file system, which can reduce overhead by reducing memory copy and switching between user mode and kernel mode. Data and metadata are stored in the server's NVM in devdax namespace (DEVDAX), and the storage device is divided into three areas in RIMFS:

(1) **Data Zone** is used to store data, and clients can directly access the data in *Data Zone*. We design memory management based on FIFO to manage *Data Zone*.
(2) **Metadata Zone** is used to store metadata. We design a file management strategy consisting of two structures to manage the metadata: tree structure is used to manage all file metadata, and hash structure is used to optimize the query. This design can solve the problem of low query efficiency of the tree structure and write magnification caused by the hash structure during modification.
(3) **Super block** is the entry point for system recovery.

Data access: The data access of RIMFS is designed based on the client-active [16] pattern. The metadata cache on the client side caches a part of the historical access metadata, and RIMFS achieves low access overhead by reducing metadata request operations and using the one-side operations of RDMA as much as possible. For small-grained IO requests, the cost of fetching metadata affects IO performance. Therefore, the design of metadata caching and prefetching is necessary to reduce the cost mentioned above. As shown in the Figure 8, the communication process of RIMFS read is divided into three steps: (1) client sends metadata request information to server; (2) server returns metadata information; (3) client initiates *RDMA read* to read data in server's *Data Zone*. When metadata cache hits, steps (1) and (2) of the above process would be unnecessary, and client directly initiates *RDMA read* to read data.

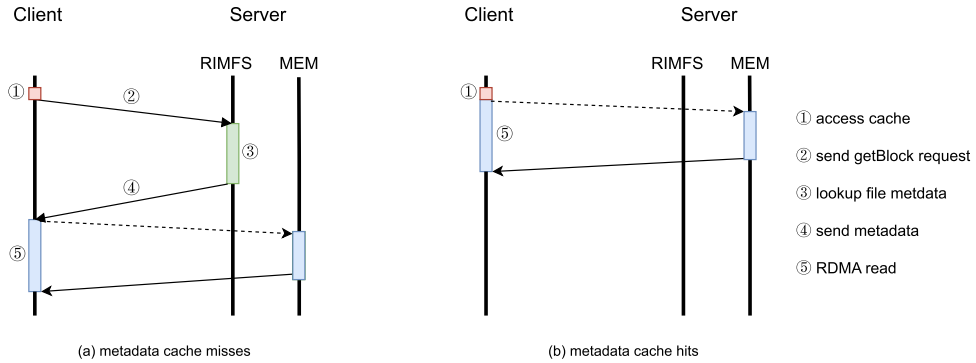


Fig. 8. The flow chart of RIMFS read. Figure (a) shows the flow chart of read when the metadata cache misses; Figure (b) shows the flow chart of read when the metadata cache hits.

6.2 RIMFS Experimental Setup

The configuration details of the cluster used in the experiments as shown in Section 4. We use local Ext4-DAX (Ext4-DAX_L) and Ext4-DAX over RDMA (Ext4-DAX_R) to compare with RIMFS. All file systems are deployed on the server's NVM. Ext4-DAX [34, 35] is a kernel file system designed for NVM that is optimized to bypass page cache and reduce memory copies. Ext4-DAX over RDMA uses the NFS [36] protocol that supports RDMA to allow clients to directly access the Ext4-DAX file system mounted on the server over RDMA.

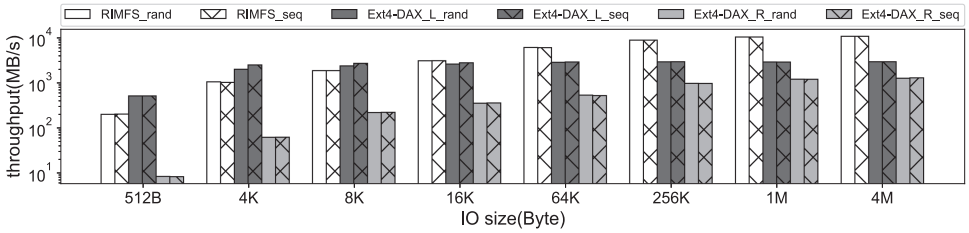
To evaluate the read/write performance of file systems, a simulated HDFS [1] micro-benchmark is used. To test the synthetic performance of file systems, several workloads of filebench including fileserver, webserver, varmail, mongo, and copyfile are also considered. Besides, to reflect the overhead of file systems, the throughput of the read/write under different CPU utilization is evaluated, in which the CPU utilizations are 0% ~ 90%.

6.3 Micro-benchmark

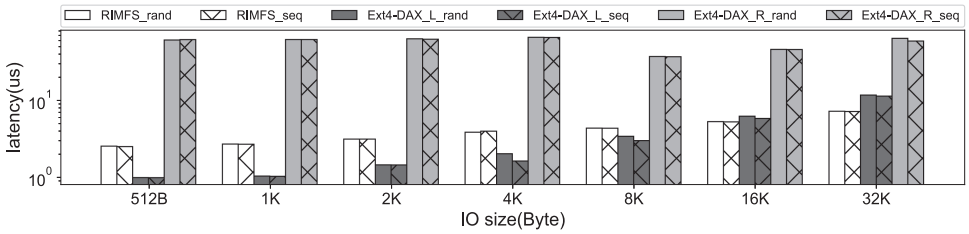
In this section, the random/sequential read/write performance of different file systems with different IO sizes are evaluated. As shown in Figure 9 and Figure 10, for RIMFS, Ext4-DAX_L, and Ext4-DAX_R, random read/write will achieve the same performance as sequential read/write. The reason is that the page cache, which causes random read/write performing worse than sequential read/write, is avoided in RIMFS and Ext4-DAX. So there is no significant difference between sequential and random read/write in performance.

6.3.1 Results and Analysis for Read. Figure 9 shows the performances of evaluated file systems in terms of the read latency and read throughput by changing the read sizes. The Ext4-DAX_R performs worst in all situations compared with the other two file systems. The throughput of Ext4-DAX_L increases stably before 8 KB and is higher than RIMFS, however, it reaches a peak and increases negligibly after that. At the same time, the throughput increase of RIMFS is rapid continuously, and the throughput of RIMFS exceeds that of Ext4-DAX_L after 8 KB.

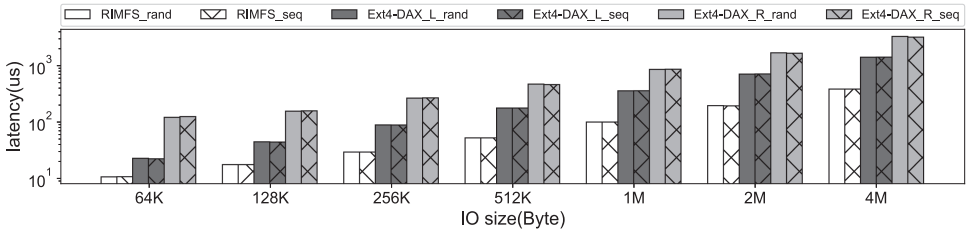
When the **IO size is smaller than 8 KB**, the read latency of RIMFS is higher than that of Ext4-DAX_L, specifically, the read latency of RIMFS is 27.6% higher than that of Ext4-DAX_L on 8 KB IO size. This is **because** when the IO size is small, the overhead of network transmission is more obvious, and the read latency of RIMFS will be higher than that of Ext4-DAX_L that does not need network transmission. However, with the increase of IO size, the read latency of Ext4-DAX_L increases obviously, while it increases much slowly for RIMFS.



(a) Throughput of read



(b) Latency of read



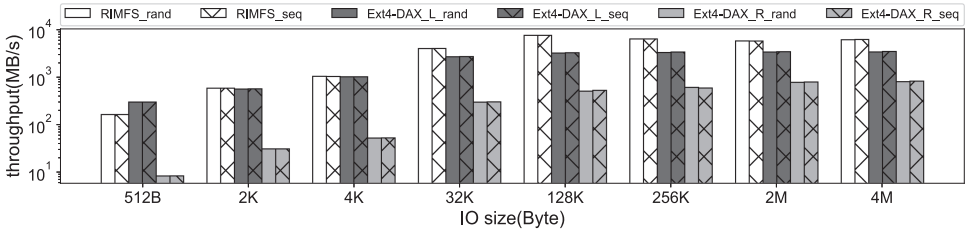
(c) Latency of read

Fig. 9. Comparison of the latency and throughput of sequential/random read. The latency of Ext4-DAX_L is lower than RIMFS on small IO size, but the latency of Ext4-DAX_L increases rapidly and exceeds the latency of RIMFS. The throughput of RIMFS is lower than that of Ext4-DAX_L when the IO size is smaller than 8 KB, and exceeds it as the IO size is larger than 8 KB.

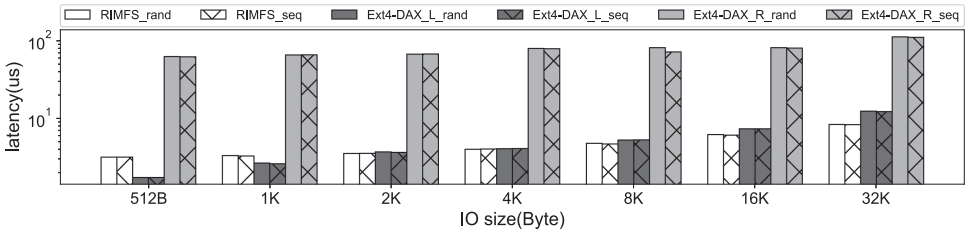
When the **IO size is larger than 8 KB**, the read latency of RIMFS is lower than that of Ext4-DAX_L. When the IO size exceeds 32 KB, the throughput of Ext4-DAX_L reaches its bandwidth limit, while the throughput of RIMFS increases rapidly. This is **because** using RDMA to access NVM allows NVM to perform better than local access in the case of large IO size, as described in Section 5. Specifically, on 4 MB IO size, the read latency of RIMFS is 72.9% lower than that of Ext4-DAX_L, and the sequential/random read throughput of RIMFS is about 10845 MB/s and 10830 MB/s, respectively.

At the same time, **in any case of IO size**, the read latency of RIMFS is lower than that of Ext4-DAX_R, overall, the average latency of RIMFS is 90.6% lower than that of Ext4-DAX_R. The results show that it is not efficient to combine the NVM file system such as Ext4-DAX and RDMA protocol immediately as Ext4-DAX_R, in which there are more software overheads including data copy from network stack to storage stack [37]. While, in RIMFS, data are directly accessed from NVM through the RDMA NIC, which reduces many memory copies.

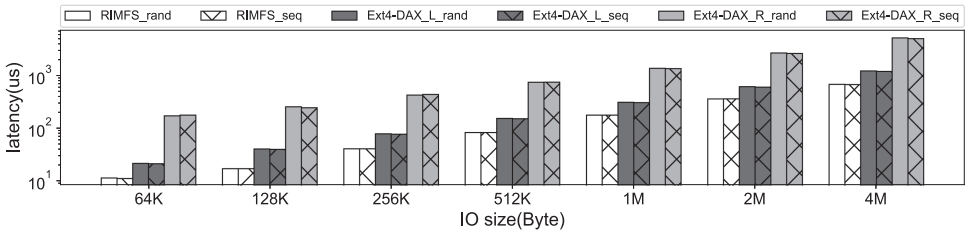
6.3.2 Results and Analysis for Write. Figure 10 shows the performance of the file system in terms of write latency and write throughput by changing the write sizes. On the whole, the performance of Ext4-DAX_R is the worst in this experiment. In **any case of IO size**, the write latency of RIMFS is lower than that of Ext4-DAX_R, overall, the average latency of RIMFS is 91.7% lower than that of



(a) Throughput of write



(b) Latency of write



(c) Latency of write

Fig. 10. Comparison of the latency and throughput of sequential/random write. The latency of Ext4-DAX_L is lower than RIMFS on small IO size. But the latency of Ext4-DAX_L increases rapidly and exceeds the latency of RIMFS. The throughput of RIMFS is lower than that of Ext4-DAX_L when the IO size is smaller than 1 KB, and exceeds it as the IO size is larger than 1 KB.

Ext4-DAX_R and the latency of RIMFS is almost lower than Ext4-DAX_L and Ext4-DAX_R. As the IO size increases, the throughput of Ext4-DAX_L increases slowly while the throughput of RIMFS increases rapidly, and both of them reach their peak when the IO size is equal to 128 KB.

The write latency of RIMFS is 24.4% higher than that of Ext4-DAX_L on **1 KB IO size**; but as the IO size exceeds 1 KB, the write latency of Ext4-DAX_L exceeds that of RIMFS. The **reason** for this result has been mentioned above. It is worth noting that only when **the IO size is smaller than 1 KB**, the write latency of Ext4-DAX_L is lower than that of RIMFS; while the read latency of EXt-DAX_L will exceed that of RIMFS after the IO size is higher than 8 KB. It is **because** the write latency of the NVM is higher than the read latency, so the network overhead has less impact on the write latency than on the read latency.

When the IO size is larger than 1 KB, the write latency of RIMFS is lower than that of Ext4-DAX_L. Specifically, on 128 KB IO size, the write throughput of RIMFS and Ext4-DAX_L reach their peak, and the write latency of RIMFS is 57.7% lower than that of Ext4-DAX_L, and the sequential/random write throughput of RIMFS is about 7405 MB/s and 7385 MB/s, respectively.

6.4 Macro-benchmark

In this section, different workloads from filebench are tested in different file systems to evaluate the overall performances of them. Figure 11 measures the I/O throughput for handling concurrent

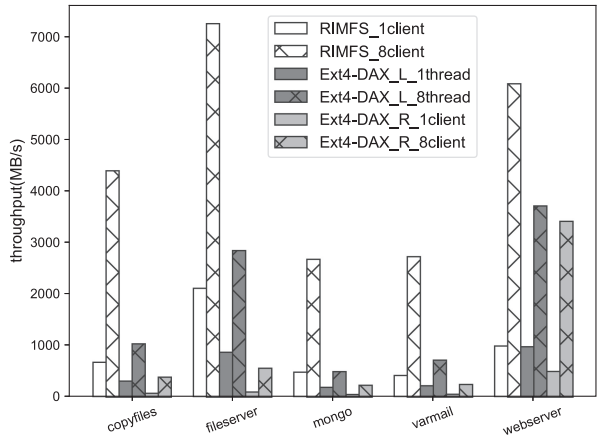


Fig. 11. Comparison of the throughput in different workloads. RIMFS exhibits good scalability with rising client counts under all workloads.

requests from different numbers of clients/threads in RIMFS, Ext4-DAX_L, and Ext4-DAX_R based on the workloads of copyfile, fileserver, mongo, varmail, and webserver of filebench.

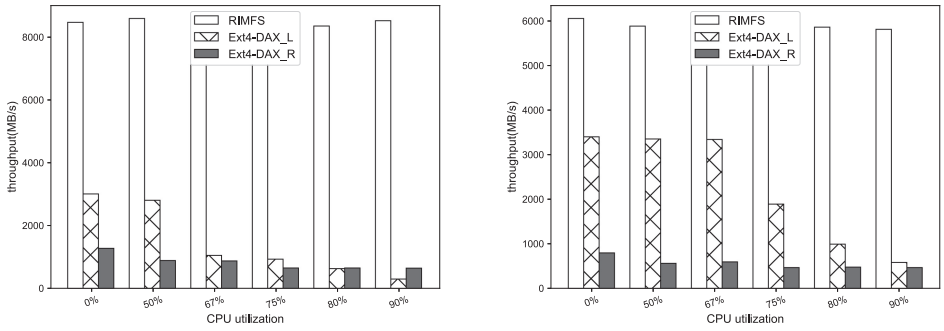
We evaluate the request processing overhead and request processing ability of different file systems by increasing the number of clients/threads to increase the CPU load. As shown in Figure 11, as the number of the client requests increasing, RIMFS throughput increases significantly compared with the other file systems, which **reflects** that when the CPU load increases, RIMFS requires less CPU resources when processing requests and handles more requests.

Specifically, under the varmail workload, with 8 clients/threads, RIMFS performs 2725 MB/s throughput - 6.5 × the single-client performance; Ext4-DAX_L performs 717 MB/s throughput - 2.61 × the single-thread performance; and Ext4-DAX_R performs 245 MB/s throughput - 4.43 × the single-client performance. This **reflects** the greater demand for CPU resources from Ext4-DAX_L and Ext4-DAX_R. As a result, when the number of clients/threads increases, Ext4-DAX_L and Ext4-DAX_R cannot process multiplied requests in parallel, while RIMFS will achieve relatively higher performance improvement. Meanwhile, with 8 clients/threads, the throughput of RIMFS is 2.8 × higher than that of Ext4-DAX_L, and 10.12 × higher than that of Ext4-DAX_R. And under the workloads of copyfiles and mongo, they show a consistent trend as the varmail workload.

Under the fileserver workload, with 8 clients/threads, the throughput of RIMFS is 7248 MB/s which is 3.43 × the single-client's throughput; the throughput of Ext4-DAX_L is 844 MB/s which is 3.26 × the single-thread's throughput; and the throughput of Ext4-DAX_R is 560 MB/s which is 5.66 × the single-client's throughput. We notice that the improvement of Ext4-DAX_L is more than that of RIMFS. The **reason** is that the type of fileserver workload is **data IO-intensive**, in which I/O requests from RIMFS have taken up the bandwidth of the server's NVM, resulting in an increase in latency for each I/O request; however, Ext4-DAX_R is far from reaching the bandwidth limit - the bandwidth of Ext4-DAX_R is only 560 MB/s with 8 clients. What's more, with 8 clients/threads, the throughput of RIMFS is 2.55 × higher than that of Ext4-DAX_L, and 11.94 × higher than that of Ext4-DAX_R. And under the webserver workload, they show a consistent trend as the fileserver workload.

6.5 CPU Non-sensitive

To illustrate the low CPU requirement of RIMFS, we evaluate the sequential read and write throughput of RIMFS, Ext4-DAX_L and Ext4-DAX_R under different server's CPU utilization from 0% to



(a) read (b) write

Fig. 12. Comparison of the read/write throughput under different CPU utilization. The read/write throughput remains almost unchanged whatever the CPU utilization is. But the throughput of Ext4-DAX_L and Ext4-DAX_R drop with the CPU utilization increasing.

90%. As shown in Figure 12, as the CPU utilization increases, the read/write throughput of RIMFS remains almost unchanged. And when the CPU utilization is low, the read/write throughput of Ext4-DAX_L is relatively high; however, when the CPU utilization exceeds a certain percentage, the throughput drops rapidly.

For Ext4-DAX_R, the read/write throughput is almost the lowest significantly, and with the increasing of CPU utilization, the throughput decreases steadily. It **reflects** that both CPU capacity and software layers contribute to the overhead of Ext4-DAX_R, and the influence of software layers is explained in Section 6.3.

For Ext4-DAX_L, (1) when the CPU utilization is lower than 50%, the read throughput is kept at about 3005 MB/s; (2) as the CPU utilization increases, the read throughput decreases rapidly; (3) when the CPU utilization is set to 90%, the read throughput is only 299 MB/s, which is lower than that of Ext4-DAX_R. (4) the write throughput decreases rapidly after the CPU utilization exceeds 67%. The **reason** is that there are multiple data copies on the read/write I/O path of Ext4-DAX_L, which requires CPU participation and causes high overhead. Therefore, when the CPU utilization is too high, the CPU resources required for the read/write are preempted. As a result, the read/write throughput drops rapidly.

For RIMFS, (1) regardless of the CPU utilization, its read/write throughput remains almost unchanged, and is kept at about 8507 MB/s and 5849 MB/s, respectively. (2) when the CPU utilization is 90%, the read throughput of RIMFS is 8507 MB/s, which is 27.41 × and 12.14 × higher than that of Ext4-DAX_L and Ext4-DAX_R, respectively. The **reason** is that the I/O path of RIMFS will not need to pass through the CPU, and the data is directly transferred between the RDMA NIC and the NVM. Therefore, the increase in CPU utilization will not affect the throughput of data transmission.

7 CONCLUSION

This paper explores and analyzes the architectures based on NVM over RDMA comprehensively. By comparing RDMA accessing FSDAX with DEVDAX, an important conclusion is obtained: the performance of RDMA accessing DEVDAX is better than that of FSDAX. And to further explore the highest performance that can be achieved by the architecture of NVM over RDMA, we compare DEVDAX over RDMA with several architectures, including DRAM over RDMA, local DEVDAX, and local DRAM. Then we get the following conclusions: (1) the performance of DEVDAX over RDMA is close to that of DRAM over RDMA; (2) in the case of large IO size, the performance of

DEVDAX over RDMA is better than that of local DEVDAX; (3) compared with local architectures, the performance improvement of DEVDAX over RDMA is greater than that of DRAM over RDMA. Finally, based on the architecture of DEVDAX over RDMA, we design a file system prototype RIMFS with low overhead, low latency, high throughput, and low CPU requirements. Experimental results show that RIMFS has a higher peak performance than Ext4-DAX with lower software overhead: the read throughput of RIMFS is 268.4% higher than that of local Ext4-DAX on 4 MB IO size. And when the CPU utilization is 90%, the performance of RIMFS is hardly reduced, while the performances of local Ext4-DAX and Ext4-DAX over RDMA are reduced significantly. For example, the read throughput of RIMFS is 8507 MB/s, which is 27.41 × higher than that of local Ext4-DAX and 12.14 × higher than that of Ext4-DAX over RDMA. In the future, we will improve the functions of RIMFS, such as consistency and data integrity.

ACKNOWLEDGMENTS

This work is partially supported by NSFC 61972154 and Shanghai Science and Technology Commission Project 20511101600.

REFERENCES

[1] Konstantin Shvachko, Hairong Kuang, Sanjay Radia, and Robert Chansler. 2010. The hadoop distributed file system. In *2010 IEEE 26th symposium on mass storage systems and technologies (MSST)*. IEEE, 1–10.

[2] Dhruba Borthakur. 2007. The hadoop distributed file system: Architecture and design. *Hadoop Project Website* 11, 2007 (2007), 21.

[3] Sage A. Weil, Scott A. Brandt, Ethan L. Miller, Darrell D. E. Long, and Carlos Maltzahn. 2006. Ceph: A scalable, high-performance distributed file system. In *Proceedings of the 7th symposium on Operating systems design and implementation*. 307–320.

[4] Asif Ali Khan, Andrés Goens, Fazal Hameed, and Jeronimo Castrillon. 2020. Generalized data placement strategies for racetrack memories. In *2020 Design, Automation & Test in Europe Conference & Exhibition (DATE)*. IEEE, 1502–1507.

[5] Rui Xu, H.-M. Sha Edwin, Qingfeng Zhuge, Shouzhen Gu, and Liang Shi. 2020. Optimizing data placement for hybrid SPM with SRAM and racetrack memory. In *The 38th IEEE International Conference on Computer Design (ICCD)*. ACM.

[6] Moinuddin K. Qureshi, Vijayalakshmi Srinivasan, and Jude A. Rivers. 2009. Scalable high performance main memory system using phase-change memory technology. In *Proceedings of the 36th annual international symposium on Computer architecture*. 24–33.

[7] Dmitri B. Strukov, Gregory S. Snider, Duncan R. Stewart, and R. Stanley Williams. 2008. The missing memristor found. *nature* 453, 7191 (2008), 80–83.

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

[9] Jian Yang, Juno Kim, Morteza Hoseinzadeh, Joseph Izraelevitz, and Steve Swanson. 2020. An empirical guide to the behavior and use of scalable persistent memory. In *18th {USENIX} Conference on File and Storage Technologies ({FAST} 20)*. 169–182.

[10] Tony Mason, Thaleia Dimitra Doudali, Margo Seltzer, and Ada Gavrilovska. 2020. Unexpected performance of intel® optane dc persistent memory. *IEEE Computer Architecture Letters* 19, 1 (2020), 55–58.

[11] Edwin H.-M. Sha, Xianzhang Chen, Qingfeng Zhuge, Liang Shi, and Weiwen Jiang. 2016. A new design of in-memory file system based on file virtual address framework. *IEEE Trans. Comput.* 65, 10 (2016), 2959–2972.

[12] Subramanya R. Dulloor, Sanjay Kumar, Anil Keshavamurthy, Philip Lantz, Dheeraj Reddy, Rajesh Sankaran, and Jeff Jackson. 2014. System software for persistent memory. In *Proceedings of the Ninth European Conference on Computer Systems*. 1–15.

[13] Xiaojian Wu and A. L. Narasimha Reddy. 2011. SCMFS: A file system for storage class memory. In *Proceedings of 2011 International Conference for High Performance Computing, Networking, Storage and Analysis*. 1–11.

[14] Youngjin Kwon, Henrique Fingler, Tyler Hunt, Simon Peter, Emmett Witchel, and Thomas Anderson. 2017. Strata: A cross media file system. In *Proceedings of the 26th Symposium on Operating Systems Principles*. 460–477.

[15] Mellanox Technologies. RDMA Aware Networks Programming User Manual. Retrieved from https://docs.mellanox.com/display/RDMAAwareProgrammingv17/RDMA+Aware+Networks+Programming+User+Manual.

[16] Youyou Lu, Jiwu Shu, Youmin Chen, and Tao Li. 2017. Octopus: an rdma-enabled distributed persistent memory file system. In *2017 {USENIX} Annual Technical Conference ({USENIX} {ATC} 17)*. 773–785.

[17] Jian Yang, Joseph Izraelevitz, and Steven Swanson. 2019. 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)*. 221–234.
[18] Nusrat Sharmin Islam, Md Wasi-ur Rahman, Xiaoyi Lu, and Dhabaleswar K. Panda. 2016. High performance design for HDFS with byte-addressability of NVM and RDMA. In *Proceedings of the 2016 International Conference on Supercomputing*. 1–14.
[19] Brian Beeler. Intel Optane DC Persistent Memory Module (PMM). Retrieved from https://www.storagereview.com/news/intel-optane-dc-persistent-memory-module-pmm.
[20] Mellanox Technologies. ConnectX-6 VPI Card. Retrieved from https://docs.mellanox.com/display/ConnectX6VPI.
[21] Intel Corporation. Persistent Memory Replication Over Traditional RDMA. Retrieved from https://software.intel.com/content/www/us/en/develop/articles/persistent-memory-replication-over-traditional-rdma-part-1-understanding-remote-persistent.html#Durability.
[22] Intel Corporation. Intel® Data Direct I/O Technology. Retrieved from https://www.intel.com/content/www/us/en/io/data-direct-i-o-technology.html.
[23] Aleksandar Dragojević, Dushyanth Narayanan, Miguel Castro, and Orion Hodson. 2014. FaRM: Fast remote memory. In *11th {USENIX} Symposium on Networked Systems Design and Implementation ({NSDI} 14)*. 401–414.
[24] Jithin Jose, Hari Subramoni, Miao Luo, Minjia Zhang, Jian Huang, Md Wasi-ur Rahman, Nusrat S. Islam, Xiangyong Ouyang, Hao Wang, Sayantan Sur, et al. 2011. Memcached design on high performance rdma capable interconnects. In *2011 International Conference on Parallel Processing*. IEEE, 743–752.
[25] Jithin Jose, Hari Subramoni, Krishna Kandalla, Md Wasi-ur Rahman, Hao Wang, Sundeep Narravula, and Dhabaleswar K. Panda. 2012. Scalable memcached design for infiniband clusters using hybrid transports. In *2012 12th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing (ccgrid 2012)*. IEEE, 236–243.
[26] Jaehyun Hwang, Qizhe Cai, Ao Tang, and Rachit Agarwal. 2020. {TCP} ≈ {RDMA}: CPU-efficient remote storage access with i10. In *17th {USENIX} Symposium on Networked Systems Design and Implementation ({NSDI} 20)*. 127–140.
[27] Anuj Kalia, David Andersen, and Michael Kaminsky. 2020. Challenges and solutions for fast remote persistent memory access. In *Proceedings of the 11th ACM Symposium on Cloud Computing*. 105–119.
[28] Patrick Stuedi, Animesh Trivedi, Jonas Pfefferle, Radu Stoica, Bernard Metzler, Nikolas Ioannou, and Ioannis Koltsidas. 2017. Crail: A high-performance I/O architecture for distributed data processing. *IEEE Data Eng. Bull.* 40, 1 (2017), 38–49.
[29] Patrick Stuedi, Animesh Trivedi, Jonas Pfefferle, Ana Klimovic, Adrian Schuepbach, and Bernard Metzler. 2019. Unification of temporary storage in the nodekernel architecture. In *2019 {USENIX} Annual Technical Conference ({USENIX}{ATC} 19)*. 767–782.
[30] Patrick Stuedi, Animesh Trivedi, Bernard Metzler, and Jonas Pfefferle. 2014. Darpc: Data center rpc. In *Proceedings of the ACM Symposium on Cloud Computing*. 1–13.
[31] Yiying Zhang, Jian Yang, Amirsaman Memaripour, and Steven Swanson. 2015. 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*. 3–18.
[32] Jiwu Shu, Youmin Chen, Qing Wang, Bohong Zhu, Junru Li, and Youyou Lu. 2020. TH-DPMS: Design and implementation of an rdma-enabled distributed persistent memory storage system. *ACM Transactions on Storage (TOS)* 16, 4 (2020), 1–31.
[33] Linux Kernel Organization. Direct Access for files. Retrieved from https://www.kernel.org/doc/Documentation/filesystems/dax.txt.
[34] Matthew Wilcox. Add support for NV-DIMMs to ext4. Retrieved from https://lwn.net/Articles/613384/.
[35] Matt DeBergalis, Peter Corbett, Steve Kleiman, Arthur Lent, Dave Noveck, Tom Talpey, and Mark Wittle. 2003. The direct access file system. (2003).
[36] Brent Callaghan, Theresa Lingutla-Raj, Alex Chiu, Peter Staubach, and Omer Asad. 2003. Nfs over rdma. In *Proceedings of the ACM SIGCOMM workshop on Network-I/O convergence: experience, lessons, implications*. 196–208.
[37] Jian Yang, Joseph Izraelevitz, and Steven Swanson. 2020. FileMR: Rethinking {RDMA} Networking for Scalable Persistent Memory. In *17th {USENIX} Symposium on Networked Systems Design and Implementation ({NSDI} 20)*. 111–125.

Received April 2021; revised June 2021; accepted July 2021