Alleviating Performance Interference Through Intra-Queue I/O Isolation for NVMe-over-Fabrics

Wenhao Gu, Xuchao Xie and Dezun Dong

November 28, 2022

Alleviating Performance Interference through Intra-Queue I/O Isolation for NVMe-over-Fabrics

Wenhao Gu[1], Xuchao Xie[1], and Dezun Dong[⋆]

College of Computer, National University of Defense Technology, Changsha 410073, China
{guwenhao16,xiexuchao,dong}@nudt.edu.cn

Abstract. The NVMe-over-Fabrics (NVMeoF) protocol enables high-performance Protocol Data Units (PDUs) exchanges between hosts and remote NVMe controllers. The performance benefits of NVMeoF are mainly derived from the multiple deep queue pairs for parallel PDUs transfers. NVMeoF has significantly facilitated NVMe SSD disaggregation from compute nodes for better resource utilization and scaling independence. However, as the performance of NVMe SSD and network infrastructure increases, the near-perfect performance delivery of NVMeoF is harder to achieve. The primary reason is the increased CPU interrupts and performance interference originated from the I/O requests served by the same NVMeoF queue pair.
In this paper, we investigate how intra-queue requests are mutually affected, and propose PINoF, a Performance Isolated remote storage access mechanism for NVMe-over-Fabrics. PINoF separates CMD and Data PDUs resources in each NVMeoF queue pair to achieve intra-queue I/O isolation, transfers PDUs in batch along with read or write specific I/O path to achieve isolated interrupt-coalescing, and introduces differentiated PDU reordering schemes to achieve isolated scheduling. Our experimental results demonstrate that compared with state-of-the-art NVMeoF implementations, PINoF achieves 23.92% lower latency, increases bandwidth by up to 19.59%, and improves IOPS by 12.41% on average.

Keywords: NVMeoF · Performance Interference · I/O Isolation.

1 Introduction

Resource disaggregation architecture has been significantly facilitated by recent advances in high-speed network technologies[7]. Meanwhile, various resource-specific interconnection protocols have been developing for ultra-low latency and high throughput communication between different kinds of disaggregated resources. In terms of storage disaggregation, as NVMe SSDs perform much faster than SAS/SATA SSDs and Hard Disk Drives (HDDs)[20], the software overheads in the I/O path to NVMe SSDs become much more pronounced[2]. Recently, the NVMe-over-Fabrics (NVMeoF) protocol[15] reclaims that it can dramatically reduce network and processing overheads[8], thus achieving negligible performance degradation for remote storage access through RDMA, FC,

[1] Equal contribution
[⋆] Corresponding author

and TCP networks[6]. Compared with the iSCSI protocol that was originally designed for disaggregating HDDs, the performance benefits of NVMeoF are mainly derived from the multiple deep queue pairs for parallel PDUs (Protocol Data Units) transfers that can fully utilize the internal parallelism of SSDs[18].

Unlike NVMe-over-RDMA that builds on the basis of large-scale RDMA-enabled specific network infrastructures, NVMe-over-TCP (NoT) is a recent transport extension of NVMeoF that can be implemented on top of commodity Ethernet hardware and standard TCP/IP protocol stack[11, 17]. NoT promotes NVMeoF deployment scenarios to the most common network infrastructure in datacenters without building out separate storage networks. Specifically, NoT defines how NVMe command (CMD) and data are encapsulated within TCP PDUs and provides regulations of how the queue pairs of host and remote NVMe controller are mapped to TCP connections and CPU cores.

As NoT inherits the superiorities of both NVMe and TCP/IP protocols, it comes at a cost of the flaws of TCP/IP software stack, such as additional memory copies and more CPU overhead compared with NVMe-over-RDMA[4, 5, 13]. Therefore, as the performance of NVMe SSD and network infrastructure increases, NoT has to cost more CPU processing overhead on both host and target sides to achieve full performance delivery of NVMe SSDs. The primary reason is the increased CPU interrupts and performance interference originated from the CMD and Data PDUs served by the same queue pair[1]. Hwang et al. have proposed i10[9, 10] to delay ringing doorbells to accumulate requests and process the requests in batch to amortize network and software overhead. However, accumulating requests further exacerbates performance interference issues of the PDUs processed in the same batch.

As the performance interference issue caused by the diverse PDUs of requests in NoT implementation has not been paid sufficient attention and is rarely studied, in this work, we investigate how intra-queue requests are mutually affected in current NoT implementation and propose PINoF, a Performance Isolated remote storage access mechanism for NVMe-over-Fabrics. Overall, our main contributions in this paper can be summarized in three aspects as below.

• Intra-Queue I/O Isolation. We propose to separate data structures and resources of CMD and Data PDUs in each PINoF queue pair to isolate the PDUs generated by read and write I/O, thus PINoF can greatly mitigate the performance interference between CMD and Data PDUs and manage CMD and Data PDUs with the same kind of operation independently.

• Specific I/O Paths. PINoF always transfers NoT PDUs in batch along with specific read or write I/O path that uses specially designed interrupt-coalescing strategy, thus the performance interference among the PDUs generated from the requests processed in a same batch can be further mitigated.

• Isolated Scheduling. PINoF further introduces differentiated PDU reordering schemes by offering different priorities for different types of PDUs. In this case, the waiting latency of the PDUs that potentially affect subsequent PDUs can be significantly reduced.

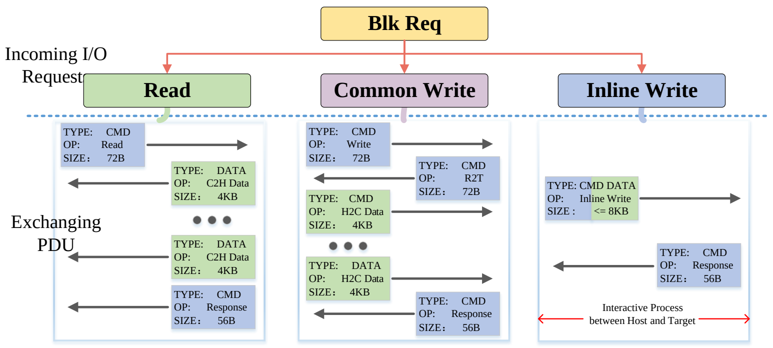

Blk Req
Incoming I/O Request
Read
Common Write
Inline Write
Exchanging PDU
TYPE: CMD OP: Read SIZE: 72B
TYPE: DATA OP: C2H Data SIZE: 4KB
TYPE: CMD OP: Response SIZE: 56B
TYPE: CMD OP: Write SIZE: 72B
TYPE: CMD OP: R2T SIZE: 72B
TYPE: CMD OP: H2C Data SIZE: 4KB
TYPE: DATA OP: H2C Data SIZE: 4KB
TYPE: CMD DATA OP: Inline Write SIZE: <= 8KB
Interactive Process between Host and Target

Fig. 1. NoT PDUs generated by the I/O requests with different types and sizes

We implemented PINoF in Linux kernel and evaluated it by FIO and RocksDB benchmarks. Our experimental results demonstrate that compared with state-of-the-art NVMeoF implementations, PINoF can achieve 23.92% lower latency, increase bandwidth by up to 19.59%, and improve IOPS by 12.41% on average.

2 Background and Motivation

2.1 NVMe-over-TCP

NoT host needs to establish a connection to remote NVMe controller in target to enable transfers. The process of connection is to create multiple one-to-one mappings between host queues and controller queues. Each host queue and its associated controller queue will be mapped to a specific TCP connection and a separate CPU core. As long as a NoT connection is established, NoT drivers will encapsulate the NVMe command, response, and data into TCP PDUs, transferred along standard TCP/IP protocol stack[14, 19]. Generally, there are five kinds of PDUs used in NoT implementation, i.e., Read/Write CMD PDU, R2T (Ready to Transfer Command) PDU, H2CData (Host to Controller Data) PDU, C2HData (Controller to Host Data) PDU, and Resp (Response) PDU.

The detailed NoT workflows of the I/O requests with different types and sizes can be characterized in Fig. 1. Take common write I/O as an example, for the write request that is larger than 8KB, its data to write has to be transferred by at least two H2CData PDUs. Specifically, when target receives a CMD PDU of a large-size write request, target driver will determine the size of data it can receive in the next transfer and pass the information to host via R2T PDU. Host driver will always send H2CData PDUs following the requirement in R2T PDUs. Target driver will execute the write request only if all the data to write has been transferred by H2CData PDUs. The Resp PDU will be sent to host when the write request is executed successfully on target side.

2.2 Intra-Queue Performance Interference

In this paper, we further investigate the performance interference issue[12, 16, 3] in NoT from a microscopic point of view. The performance interference mainly

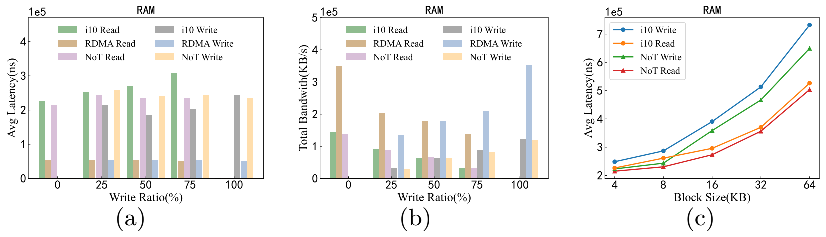


Fig. 2. Intra-queue performance interference in NVMe over TCP and i10.

comes from the I/O requests with different types and sizes that are handled in a same NoT queue. These I/O requests will finally generate a large number of NoT PDUs with different attributes in terms of types, sizes, relevance, and urgencies. Thus the common unified transceiver strategy should be carefully reconsidered, especially when multiple PDUs are accumulated for transferring in batch.

For a read request, NoT host first sends a CMD PDU of 72B to target, then NoT target returns many C2HData PDUs that contains data typically 4KB. In the case of an application with read-intensive I/O workloads, such as reading data sets from remote NVMe SSDs for machine learning, both NoT and i10 hosts will continuously process PDUs with short flow data while targets always return PDUs with long flow data. Apparently, PDUs on the host side are latency-sensitive and need to be sent out in time for processing. Meanwhile, PDUs on the target side are more suitable for gathering and transferring in batch to achieve high throughput. However, the desired PDU sending strategies of read and write requests are diametrically opposite as Fig. 1. Therefore, once these PDUs with different desires are accumulated together as i10 does, both the read and write requests have to endure severer intra-queue performance interference caused by their uncompromising PDUs.

As read and write mixed I/O workload is common in production storage systems, simply designing different NoT PDU accumulation and sending strategies on host and target sides can hardly adapt to the dynamic needs of read and write requests, and the performance interference is inevitable. As shown in Fig. 2, the read/write latency and bandwidth performance of NVMe-over-RDMA keeps stable and basically proportional respectively with the read/write ratio, while the performances of NoT and i10 are significantly interfered especially when read and write mixed. Thus, managing PDUs from read and write requests separately on both host and target sides is essential to accurately provide the most appropriate PDU accumulation and sending strategies.

Besides type and size, NoT PDUs may present significant differences in relevance and urgency. For example, even if the CMD PDU of a write request is transferred to target in time, the data of the write request cannot be transferred until its associated R2T PDU returns back. For both read and write requests, even if their data have been transferred, NoT will not signal blk-mq (multiple per-core block queues) the finish of these I/O requests since not received their Resp PDUs. Thus, R2T and Resp PDUs need to be prioritized to transfer for their relevance to H2CData PDU and urgency to finish an I/O request to further

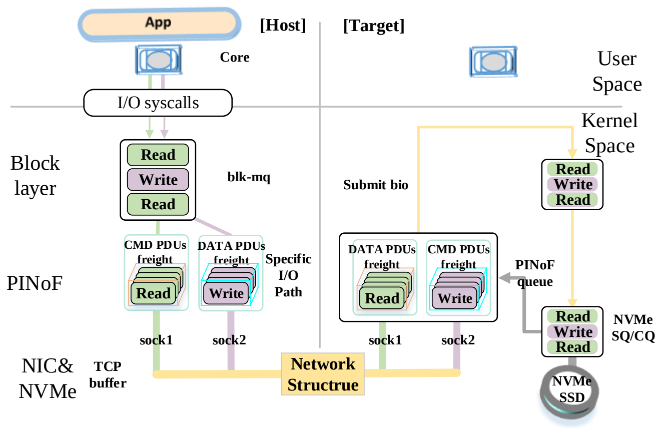


Fig. 3. System overview of PINoF.

mitigate the performance interference. As shown in Fig. 2(c), the write request latency increases significantly after the block size is 16 compared with read, due to the R2T PDU transmission.

2.3 Motivation

In this paper, we propose intra-queue I/O isolation to alleviate the performance interference in NoT. Our work is motivated by the observations below.
• Performance overhead can be amortized. Aggregating multiple requests for batch processing can amortize the overheads from network processing and TCP /IP software stack, thus the performance of NoT can be improved.
• PDU attribute should be distinguished. I/O requests with different types and sizes will generate PDUs with different attributes in terms of type, size, relevance, and urgency, which inspires differentiated PDU batch processing mechanisms.
• PDU dependency should be considered. R2T and Resp PDUs show significant relevance to H2CData PDU and urgency to finish an I/O request. Prioritizing R2T and Resp PDUs while accumulating H2CData and C2HData PDUs can achieve high throughput and mitigate I/O latency amplification simultaneously.

3 System Overview

PINoF is a modified implementation of standard NoT and works as a shim NVMe capsule and data forwarding layer between the Linux blk-mq layer and TCP/IP software stack. As shown in Fig. 3, different from standard NoT implementation, PINoF isolates both resources and strategies of CMD and Data PDUs while separating read and write requests. Specifically, the isolated resources include spatial resource “freight” and temporal resource “hrtimer”. freight is the newly established container for accumulating PDUs while hrtimer leverages the kernel high-precision timer for controlling PDU aggregation processes.

As each PINoF queue is one-to-one mapped to a hardware queue in the blk-mq layer, for each queue pair, PINoF maintains both CMD and Data PDUs

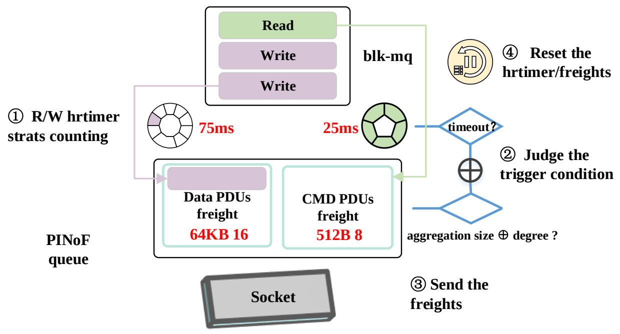


Fig. 4. Isolated resources on the host side of PINoF.

freights. The PDUs generated by an I/O request from its associated hardware queue will be distributed into either CMD or Data specific PDUs freight for aggregation by PINoF. The PDUs in the same freight have the same destination, thus can be transferred by a preassigned TCP socket. In this way, PINoF can separately manage intra-queue PDUs of read and write requests. The detailed design will be further discussed in §4.1.

The time when a freight can be sent to the remote side is cooperatively controlled by its size and the hrtimer. In PINoF, there is a trade-off between the long-time aggregation to accumulate more PDUs in a single freight and the short-time aggregation for less delays of the accumulated PDUs in the freight. To this end, PINoF separates intra-queue PDUs into different I/O paths, i.e., read specific I/O path and write specific I/O path, to achieve a good balance between interrupt-coalescing and extra delays for both read and write I/O requests. We will particularly describe the specific I/O paths in detail in §4.2.

With isolated resources and I/O paths, PINoF can achieve isolated scheduling to flexibly tune the transfer priorities of intra-queue PDUs. Besides, in PINoF, we always prioritize R2T PDUs in write specific I/O path to alleviate the PDU dependency issue, which will be discussed in §4.3.

4 PINoF Design

4.1 Intra-queue I/O Isolation

PINoF implements intra-queue I/O isolation by introducing two kinds of dedicated resources, freight and hrtimer. Both freight and hrtimer are allocated separately for the CMD and Data PDUs served by the same queue pair. Each freight is considered as a PDU container where multiple PDUs to the same destination are gathered and will be transferred along the I/O path in PINoF together. Compared with standard NoT, PINoF can significantly reduce the overhead from frequent context switches caused by fragmented PDU transmission and fully utilize the TCP acceleration technologies such as TCP segmentation offload (TSO) and generic receive offload (GRO).

As shown in Fig. 4, PINoF uses both spatial and temporal measurements to determine whether a freight should be transferred or not. Spatial measurements include aggregation size and aggregation degree. When the allocated memory size for a freight cannot accommodate the next PDU, the freight will incur an aggregation size triggered freight send operation. Similarly, a freight will incur an aggregation degree triggered send operation when the number of its accommodated PDUs exceeds the predefined threshold. PINoF introduces hrtimer as the temporal measurement of freight. Each hrtimer is used to record how much time its associated freight has spent to accumulate PDUs and has a predefined threshold to indicate the freight should be sent out. In PINoF, once a freight receives its first request, its associated hrtimer will wake up. As the hrtimer reaches its threshold, its associated freight will incur an hrtimer triggered send operation. Note that no matter whether a freight send operation is triggered by hrtimer or other spatial measurements, its hrtimer will always be reset.

4.2 Specific I/O Paths

Due to the different attributes of the PDUs generated by read and write requests, as shown in Fig. 3 and Fig. 4, the freights for read and write requests are organized and managed differently. Take read requests as an example, PINoF first prepares all the data structures needed by freights in each PINoF queue pair on both host and target sides. On host side, once a CMD PDU generated by a read request arrives, it is simply linked to the send list of the PINoF queue rather than rings the doorbell to wake up send thread directly. All the following PDUs from read requests will repeat this step until the sending condition is triggered and the freight is sent to the remote PINoF queue through socket. On the remote target side, PINoF caches freight in buffer and receives the fixed-length header to parse the header information, analyze the length of subsequent data, and receive data accordingly. This process will be repeatedly processed to realize the unpacking of the freight and parse each read request. Subsequently, the request gets off the specific I/O path, transferred to other layers for processing. The workflow of the write path is similar to that of the read, yet they mainly differ in the predetermined parameters of dedicated resources in §4.1.

kernel_sendpage() can avoid data replication on the transmission side when sending data per page, but leads to a weak batch processing capacity. While kernel_sendmsg() can copy the kernel I/O vector to the socket buffer as a function parameter, thus significantly improves the throughput at high load. Therefore, we always call kernel_sendmsg() for sending the bandwidth-intensive Data PDUs of which host write requests and target read requests mainly composed. Conversely, the CMD PDUs generated by read requests and the Resp and R2T PDUs generated by write requests are always regarded as latency-sensitive PDUs and sent by calling kernel_sendpage().

4.3 Isolated Scheduling

During the whole I/O workflow in PINoF, CMD PDUs are the initiator of all the interactions between host and target sides. Apparently, sending CMD PDUs for processing as soon as possible can significantly reduce I/O waiting latency.

As the PDUs handled on host and target sides are different, we introduce isolated scheduling mechanism in PINoF to schedule freights on host and target sides. Our basic strategy is to set different threshold for the dedicated resources of host and target. The timeout threshold of the CMD PDUs hrtimer is set to 25ms and that of the Data PDUs hrtimer is set to 75ms in PINoF. The PDUs on host side in read specific I/O path and target side in write specific I/O path are all CMD PDUs smaller than 72B, hence we heuristically set the size of CMD PDUs freight to 512B and the degree of aggregation to 8. As the Data PDUs are no less than one physical page, the Data PDUs freight size and aggregation degree are set to 64KB and 16 respectively, since 64KB is the partition upper limit of TSO technology.

Besides, we manually set the weight of R2T PDU to 3 and Resp PDU to 2 to trigger the aggregation size preferentially. Thus, the freights that contain R2T and Resp PDUs on target side will achieve highest priority to transfer in PINoF, while the freights that contain H2CData or C2HData PDUs have to endure a longer accumulation time. Consequently, the PDUs show significant relevance to H2CData PDU and urgency to finish an I/O request can be transferred in time while the H2CData and C2HData PDUs can be fully accumulated to achieve high throughput in PINoF. These parameters (high-quality setting obtained from experiments) are set loosely to prove the effectiveness of isolated scheduling.

5 Performance Evaluation

5.1 Experimental Setup

Table 1. Hardware and software configurations

	Host	**Target**
CPU	2-socket Intel (R) Xeon (R) CPU E5-2692v2@2.20GHz 12cores per socket, NUMA enabled (2 nodes)	2-socket Intel (R) Xeon (R) CPU E5-2660v2@2.60GHz 10cores per socket, NUMA enabled (2 nodes)
MEM	125GB of DRAM	64GB of DRAM
NIC	Mellanox CX-5 EX (100G) TSO/GRO=on, LRO=off, DIM disabled Jumbo frame enabled (4096B)	
SSD	N/A	1.6TB DERA D5457 NVMe SSD
IRQ	N/A	irqbalance enabled
OS	Centos 7 (kernel 5.4.43)	
FIO	version=fio–3.7, rw=randrw, size=15G cpus_allowed=0–23, runtime=300, engine=libaio iodepth=8, Direct I/O=on CPU affinity enabled, block size=4KB	

We implement PINoF as a loadable kernel module of Linux 5.4.43 by adding 723 lines of C codes on the basis of the standard NoT implementation, which can be reached at https://github.com/jackey-gu/PINoF. We build a PINoF prototype using the hardware and software configurations described in Table 1.

5.2 Evaluation Results

Performance with varying read/write ratios We evaluate the impact of intra-queue PDU isolation by setting different read/write IO request ratios in

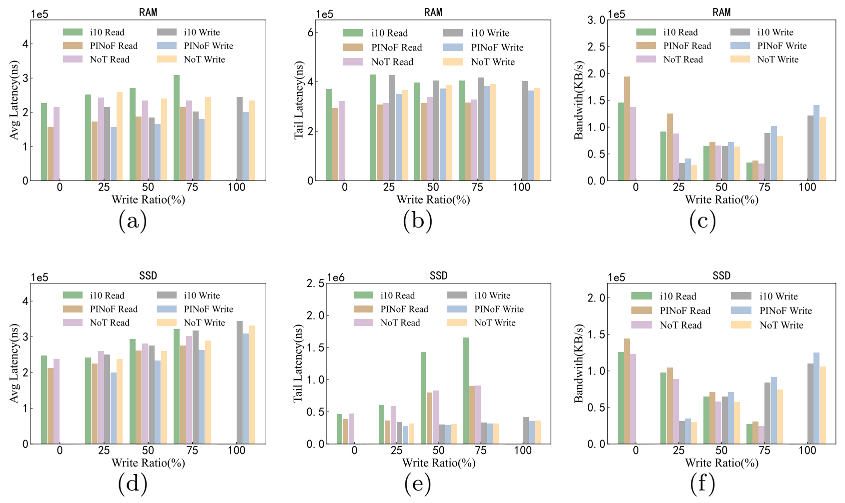


Fig. 5. Latency comparison with varying write ratio.

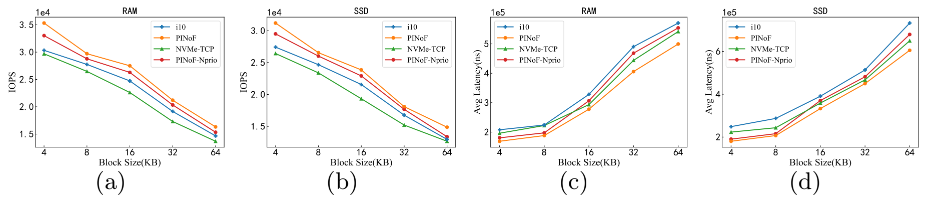


Fig. 6. IOPS and latency comparison with varying block sizes.

FIO benchmarks. As shown in Fig. 5, compared with NoT and i10, PINoF shows obvious advantages in average latency, tail latency, and bandwidth. When evaluated using RAM, compared with i10, PINoF decreases the average latency and tail latency by 31.94% and 20.45% respectively in the read-only condition. Meanwhile, PINoF provides 33.58% bandwidth growth than i10. In the write-only condition, PINoF shows about 13.52% performance improvement in terms of latency and bandwidth, which is a little bit lower than that of the read-only workload. In the condition of mixed read and write requests, PINoF presents by up to 20.57% latency reduction and 18.92% bandwidth improvement than i10.

PINoF achieves similar performance trends when evaluated using NVMe SSDs, as shown in Fig. 5. However, the overall performance improvements are lower than that of RAM. This is because the bandwidth of a single NVMe SSD can be easily saturated in the experiments, which limits the demonstration of the performance benefits of PINoF. Nevertheless, PINoF still has an 12.82% bandwidth improvement than i10 at least. As intra-queue PDU isolation can give play to the advantages of precise resource allocation and dedicated processing of read and write specific paths, it can significantly alleviate the performance interference of the aggregated PDUs no matter works alone or collaboratively with other PDU management strategies.

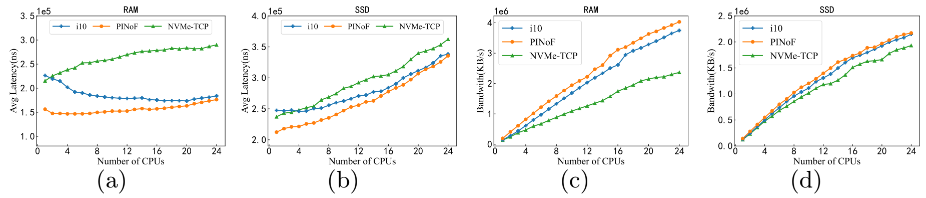


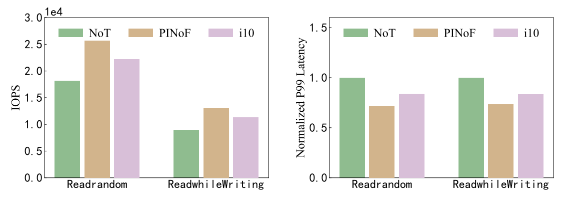
Fig. 7. Latency and bandwidth comparison with varying number of CPU cores.

Performance with varying block sizes In this experiment, we choose the write-intensive workloads with different block sizes to comprehensively evaluate the impact of the isolated scheduling design because the requests will generate a large number of R2T PDUs, which can activate the isolated scheduling in I/O path. By setting all CMD and Data PDUs aggregation conditions to 16 64KB and cancelling PDUs weight design, we enable a PINoF_Nprio system without a isolation scheduling function, so as to compare with PINoF to prove its effect.

Fig. 6 shows the IOPS and latency comparisons with varying I/O request sizes to access remote NVMe SSD and RAM block devices. Apparently, the performance of PINoF is much better than NoT and i10 no matter to access remote RAM or NVMe SSDs. PINoF can provide an IOPS improvement of 9.15% than that without designing isolated scheduling for R2T PDUs. Notably, as shown in Fig. 6(c) and 6(d), 16K block size is a dividing point of latency performance, where the trend lines of the four tested systems change dramatically. This is because when write remote storage with 4K and 8K requests, all the write requests issued from host are inline write requests that will not generate R2T PDUs. For the write requests larger than 8KB, R2T PDUs are necessary for these write requests. Besides, all the R2T PDUs are highly relevant to H2CData PDUs and considered more urgent for transferring in PINoF. In this case, the isolated scheduling design in PINoF can significantly mitigate the system performance loss caused by aggregating PDUs.

Scalability with number of cores To further understand the performance scalability of PINoF in the systems with multiple CPU cores, we evaluate the performance of PINoF with different number of CPU cores from 1 to 24. As shown in Fig. 7, both i10 and PINoF present better performance than NoT as the number of PINoF queues increases. Besides, for a fixed number of CPU cores involved in this experiment, PINoF always performs better than i10 and NoT in both latency and bandwidth. This indicates that all the designed strategies in PINoF do not incur any performance scalability loss.

As shown in Fig. 7(c), when accessing remote RAM, the bandwidth of PINoF is steadily improved as the number of CPU cores increases and is roughly proportional to the number of CPU cores. This trend does not appear in Fig. 7(d) is mainly because the performance of NVMe SSD cannot saturate the high performance of PINoF. Different from bandwidth, the average latencies of i10 and PINoF are obviously reduced compared with NoT as the number of CPU cores are used in this experiment while the average latency of PINoF increases. This is because both i10 and PINoF can benefit from the increase of CPU cores that



(a) (b)

Fig. 8. IOPS and latency comparison with RocksDB.

accelerate the trigger of threshold for batch transfer, i.e., ring the doorbell with a shorter waiting time. As the average waiting time of the PDUs aggregated in freight is shorten, PINoF finally achieves lower average I/O latency.

RocksDB Performance We use RocksDB, a widely-deployed key-value storage system, as a real application to evaluate the performance of PINoF. With Ext4 filesystem to format the SSD block device and the default db_bench tool, we populate a 43GB database containing 1,000,000,000 pieces of data.

Fig. 8 shows the evaluation results of P99 tail latency and IOPS performance. Compared with the FIO tests, due to the high application layer overhead of RocksDB, the performance improvement of PINoF slightly reduces. The throughput of PINoF is almost 0.44 times higher than that of NoT, while it is about 15.66% higher than that of i10. In terms of latency performance, since the total number of operations is fixed, the average latency is inversely proportional to IOPS, so we use P99 tail latency to represent the system latency performance. PINoF gains about a 12.91% improvement than i10, and has a greater improvement compared with NoT. As RocksDB test is to operate on the file system while FIO is to directly read the block device, thus the additional operation overhead in RocksDB leads to this performance difference. Since the application layer and filesystem occupy more latency and limit the bandwidth, the benefits obtained from the modification of the kernel software layer are partially overshadowed.

6 Conclusion

This paper introduces the design, implementation and evaluation of PINoF, a performance isolated NVMeoF design that follows the standard NoT protocol. The experimental results show that PINoF can achieve better performance in latency, bandwidth, and throughput compared with NoT and i10.

Acknowledgements We would like to thank the NPC reviewers for their insightful feedback. This work was supported in part by Excellent Youth Foundation of Hunan Province under Grant No.2021JJ10050 and the Science Foundation of NUDT under grant ZK21-03.

References

1. Ahmad, I., Gulati, A., Mashtizadeh, A.: Vic: Interrupt coalescing for virtual machine storage device io. In: 2011 USENIX Annual Technical Conference. pp. 45–58 (2011)

2. Bjørling, M., Axboe, J., Nellans, D., Bonnet, P.: Linux block io: introducing multi-queue ssd access on multi-core systems. In: Proceedings of the 6th international systems and storage conference. pp. 1–10 (2013)
3. Cheng, L., Wang, C.L.: Network performance isolation for latency-sensitive cloud applications. Future Generation Computer Systems **29**(4), 1073–1084 (2013)
4. Cobb, D., Huffman, A.: Nvm express and the pci express ssd revolution. In: Intel Developer Forum (2012)
5. Cohen, D., Talpey, T., Kanevsky, A., Cummings, U., Krause: Remote direct memory access over the converged enhanced ethernet fabric: Evaluating the options. In: 2009 17th ieee symposium on high performance interconnects. pp. 123–130 (2009)
6. Dragojević, A., Narayanan, D., Castro, M., Hodson, O.: Farm: Fast remote memory. In: 11th USENIX Symposium on Networked Systems Design and Implementation. pp. 401–414 (2014)
7. Gao, P.X., Narayan, A., Karandikar, S., Carreira, J., Han: Network requirements for resource disaggregation. In: 12th USENIX Symposium on Operating Systems Design and Implementation. pp. 249–264 (2016)
8. Guz, Z., Li, H., Shayesteh, A., Balakrishnan, V.: Performance characterization of nvme-over-fabrics storage disaggregation. ACM Transactions on Storage **14**(4), 1–18 (2018)
9. Hwang, J., Cai, Q., Tang, A., Agarwal, R.: Tcp≈rdma: Cpu-efficient remote storage access with i10. In: 17th USENIX Symposium on Networked Systems Design and Implementation. pp. 127–140 (2020)
10. Hwang, J., Vuppalapati, M., Peter, S., Agarwal, R.: Rearchitecting linux storage stack for μs latency and high throughput (2021)
11. Kaufmann, A., Stamler, T., Peter, S., Sharma: Tas: Tcp acceleration as an os service. In: Proceedings of the Fourteenth EuroSys Conference 2019. pp. 1–16 (2019)
12. Lee, M., Kang, D.H., Lee, M., Eom, Y.I.: Improving read performance by isolating multiple queues in nvme ssds. In: International Conference on Ubiquitous Information Management & Communication. p. 36 (2017)
13. Li, Y.T., Leith, D., Shorten, R.N.: Experimental evaluation of tcp protocols for high-speed networks. IEEE/ACM Transactions on networking **15**(5), 1109–1122 (2007)
14. Marinos, I., Watson, R.N., Handley, M.: Network stack specialization for performance. ACM SIGCOMM Computer Communication Review **44**(4), 175–186 (2014)
15. Minturn, D.: Nvm express over fabrics. In: 11th Annual OpenFabrics International OFS Developers' Workshop (2015)
16. Nguyen, D.T., Zhou, G., Xing, G., Qi, X., Hao, Z., Peng, G., Yang, Q.: Reducing smartphone application delay through read/write isolation. In: the 13th Annual International Conference (MobiSys 15). pp. 287–300 (2015)
17. Qiao, X., Xie, X., Xiao, L.: Load-aware transmission mechanism for nvmeof storage networks. In: International Conference on High Performance Computing and Communication (HPCCE 2021). pp. 105–112 (2022)
18. Son, Y., Kang, H., Han, H., Yeom, H.Y.: An empirical evaluation of nvm express ssd. In: International Conference on Cloud & Autonomic Computing (ICCAC 15). pp. 275–282 (2015)
19. Tai, A., Smolyar, I., Wei, M., Tsafrir, D.: Optimizing storage performance with calibrated interrupts. In: Proceedings of the 15th USENIX Symposium on Operating Systems Design and Implementation. pp. 129–145 (2021)
20. Zheng, S., Hoseinzadeh, M., Swanson, S.: Ziggurat: a tiered file system for non-volatile main memories and disks. In: 17th USENIX Conference on File and Storage Technologies. pp. 207–219 (2019)