An Evaluation of NVMe-over-Fabrics for Disaggregated Databases over Fast Networks [Short Paper]

Jigao Luo [1], Nils Boeschen [1], Tobias Ziegler [1], and Carsten Binnig [1,2]

Abstract: With increasingly faster networks, storage disaggregation in databases enhances resource utilization by independently scaling compute and storage nodes. Storage technologies like NVMe SSDs provide high throughput and low latency, but maintaining these characteristics in a disaggregated architecture poses significant challenges. NVMe-oF is a recent remote storage protocol that leverages fast network technologies such as RDMA for efficient storage disaggregation with low access overheads. In this paper, we present a first evaluation of the performance properties of NVMe-oF for database workloads to guide the development of databases. Our evaluation shows that NVMe-oF can offer high I/O performance when choosing optimal configurations for various database workloads. However, we also show that some optimizations, such as target-offload, which minimize CPU involvement on the storage side, are not (yet) working as expected.

1 Introduction

Modern networks are becoming faster. Over the past decade, networking technology has advanced significantly to meet the demands of data-centric systems and cloud-scale applications [Bi16]. Today's high-speed networks offer speeds ranging from 100 to 200 Gbps, with even faster Ethernet links—up to 400 and 800 Gbps— on the horizon. In addition to higher throughput, specialized technologies like Remote Direct Memory Access (RDMA) have seen increasing adoption, enabling efficient, high-throughput data transfers with low latency by providing direct access to remote memory. As a result, RDMA has become a critical technology for major cloud providers, contributing to its widespread use in large-scale data center networks [Ba23].

Shift toward cloud and disaggregated databases. Fast networks have enabled the shift of modern databases to the cloud, where disaggregated architectures are becoming the norm [Dr14, Dr15, Za17, Za21, Zi19, ZBL22, BJZ24]. These architectures decouple storage and compute resources, allowing them to scale independently, and are commonly used in OLAP and OLTP systems. While this disaggregation offers greater flexibility and scalability, it also increases the dependency on low-latency networks, as each storage access introduces additional network delays. As a result, there is a growing focus on leveraging fast networks to optimize storage access [Bi16, Li16].

[1] Technical University of Darmstadt, Systems Group, 64287 Darmstadt, Germany,
jigao.luo@tu-darmstadt.de, https://orcid.org/0009-0005-2263-1959;
nils.boeschen@cs.tu-darmstadt.de, https://orcid.org/0000-0001-8654-5738;
tobias.ziegler@cs.tu-darmstadt.de, https://orcid.org/0000-0002-1602-4512;
carsten.binnig@cs.tu-darmstadt.de, https://orcid.org/0000-0002-2744-7836
[2] DFKI, carsten.binnig@cs.tu-darmstadt.de, https://orcid.org/0000-0002-2744-7836

doi:10.18420/BTW2025-05

Modern SSDs: low price & high throughput. Another significant trend in recent years is the widespread replacement of magnetic disks with flash-based SSDs as database systems' primary persistent storage medium. One key factor driving this shift is cost: over the past decade, flash storage prices have dropped by 30x, with enterprise-grade SSDs now available for less than $200 per TB. Performance is another critical factor – an NVMe SSD can deliver over 3 GB/s using four PCIe 3.0 lanes, increasing to 7 GB/s with PCIe 4.0 and even 13 GB/s with PCIe 5.0. Given these advantages in both cost and performance, SSDs are expected to remain the most viable option for cost-effective storage solutions. However, the challenge of providing and accessing network-attached storage at such high bandwidths persists.
NVMe-oF: fast NVMe beyond single-node. Building on the evolutions in both modern networking and storage, NVMe-over-Fabrics (NVMe-oF) extends the NVMe protocol to enable remote storage access via fast RDMA-enabled networks. As such, NVMe-oF addresses the limitation that NVMe was originally proposed for local storage. NVMe-oF encapsulates NVMe commands and data into packets that are transmitted over networks such as RDMA, TCP, and Fiber Channel, though this paper focuses exclusively on RDMA. By utilizing RDMA, NVMe-oF offloads data movement to the network interface card (NIC), reducing the packet processing overhead.
The need for a performance study. This paper investigates how NVMe-oF can be leveraged in disaggregated database systems for different workloads (OLAP and OLTP). NVMe-oF can be deployed using two different software stacks, which are included in our study to analyze the performance trade-offs: the kernel stack and the SPDK stack. While NVMe-oF supports various block devices, this study focuses specifically on NVMe SSDs. We use I/O workloads that simulate different characteristics of OLAP and OLTP systems to evaluate performance, conducting an extensive experimental analysis. Our results show that the kernel stack faces limitations in achieving high throughput in certain configurations that are attractive for OLTP. In contrast, when properly configured, the SPDK stack consistently offers better performance across all workloads but comes with the downside that it requires ownership of the device which is not ideal in the cloud.

2 Background for NVMe-oF

2.1 NVMe-oF Basics

To leverage both high-speed networks and storage, the NVMe Working Group extended the NVMe protocol [Ex24] and proposed the NVMe-oF protocol [Ex21] for remote block devices. The NVMe-oF protocol enables direct access to remote block devices with NVMe commands, minimizing protocol conversion overhead and enhancing I/O performance. NVMe-oF shares the NVMe command set and device abstraction, including the communication with the NVMe Multi-Queue model, where NVMe commands are encapsulated in Submission Queue Entries (SQE) and NVMe completions in Completion Queue Entries (CQE).

At a high level, NVMe-oF is a protocol that exposes NVMe devices to remote clients and encapsulates NVMe commands within RDMA calls over the network. Fig. 1 illustrates

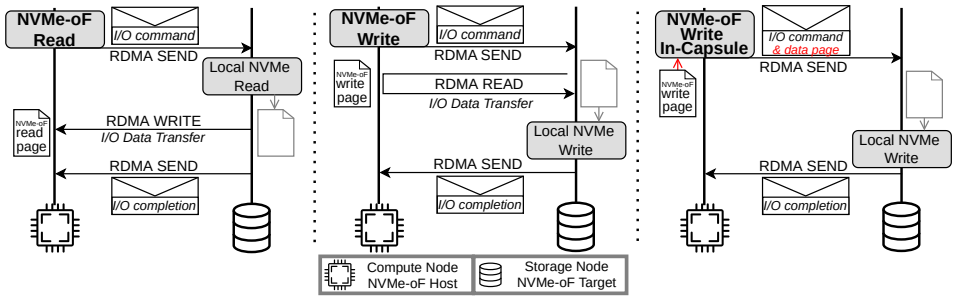

Fig. 1: A simplified flow of an I/O request (left to right): NVMe-oF read, write and write in-capsule.

the flow of an NVMe-oF I/O read from and write to a remote NVMe SSD, specifically for the annotated page. As shown, the compute node, acting as the NVMe-oF host, sends an I/O request as an NVMe command to the storage server as the NVMe-oF target. Then, the target performs I/O on its local storage and sends an I/O completion to the host. Both sides communicate using NVMe-oF capsules, which encapsulate NVMe commands and completions within RDMA_SEND payloads. In NVMe-oF reads, after receiving the command and performing the local storage read, the target transfers the retrieved page to the host using an RDMA_WRITE call. For NVMe-oF writes, the target fetches the host page via RDMA_READ and writes to its local storage. A special case is the NVMe-oF *write in-capsule*, where small payloads (typically 4 KB) are sent directly with the command, eliminating the need for an RDMA_READ.

2.2 Kernel NVMe-oF Stack

The Linux kernel provides a generalized stack for NVMe-oF. In the following, we explain the general I/O flow when using the kernel stack.

Flow from host to target. Fig. 2(a) illustrates the NVMe-oF I/O flow from the host to the target. The process begins when an application (client) on the host side issues I/O system calls via a kernel I/O interface (step 1) to access a block device mounted via NVMe-oF. Then each I/O call traverses the kernel storage stack (step 2) and is treated as a standard I/O request until it reaches the NVMe driver. Being aware of the I/O request to a remote device, the NVMe driver constructs an NVMe SQE within an NVMe-oF command capsule (step 3), which is then passed to the network stack and transmitted to the target (step 4). When capsules are received at the target (step 5), the NVMe SQEs are extracted from capsules (step 6) and a block layer request is generated from each NVMe command and submitted to the block layer (step 7). The PCIe NVMe driver receives the I/O request and submits a new NVMe SQE to the PCIe NVMe driver (step 8). The NVMe controller executes the command (step 9) and the I/O result will be returned to the host (not shown in Fig. 2(a)).

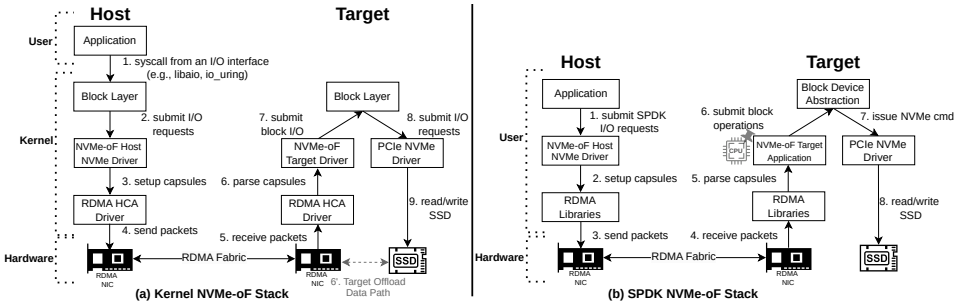

Fig. 2: NVMe-oF Stacks: Kernel and SPDK.

NVMe-oF target offload. To enhance the I/O processing efficiency on the target, NVMe-oF target offloading can be enabled. Certain NICs, such as the Mellanox ConnectX-5 [NV20], implement an NVMe-oF target offloading engine in hardware. The target offloading engine contains its on-chip hardware-based NVMe-oF target driver and PCIe NVMe driver to handle NVMe-oF commands on the data path [DC20], without incurring CPU overhead from the kernel on the target. As illustrated by step 6' (gray) in Fig. 2(a), the data flow with target offload involves only the NIC which parses NVMe-oF capsules from the network and then directly submits NVMe commands to NVMe SSDs, bypassing the kernel.

2.3 SPDK NVMe-oF Stack

An alternative to the kernel-based stack is the SPDK-based NVMe-oF stack. With the SPDK-based stack [Ya17], all handling of I/O and network is implemented in user-space, both on the host and target, which makes SPDK efficient as already known for local storage. However, the SPDK stack has limitations, as the SPDK NVMe-oF target must exclusively control the SSD, similar to the local SPDK setup, where only one client can own the SSD, preventing shared access by multiple clients. Surprisingly, this exclusive ownership restriction does not apply outside the SPDK NVMe-oF target, meaning multiple NVMe-oF hosts can share access to the same SSD provisioned by the SPDK target.

Flow from host to target. As illustrated in Fig. 2(b), I/O requests from the host application (step 1) pass through the SPDK NVMe-oF host of the NVMe driver (step 2), which utilizes lightweight RDMA libraries. Steps 3-8 follow a similar flow as the kernel stack, with the key difference being that all SPDK components are in user-space. Fig. 2(b) also illustrates that dedicated CPU cores can be pinned at the target for I/O handling. Compared to the Linux kernel stack, the SPDK stack offers three key benefits: efficient user-space drivers bypassing the kernel, using more efficient polling (for I/O and network) instead of interrupts, and lockless connection handling through dedicated core pinning. Thus, SPDK typically leads to lower latency [Gu18, ST24a] when compared to the kernel stack.

3 Experimental Evaluation

3.1 Evaluation Methodology

Access Patterns. Disaggregated database systems typically rely on specific I/O access patterns. To evaluate NVMe-oF, we simulated these patterns in our evaluation:

Point reads and writes are common in OLTP workloads and become more latency-sensitive in a disaggregated setup, network latency is introduced in each remote access between the compute and storage nodes. Latency-critical operations, such as index lookups and traversals, where prefetching is often not feasible, require random page reads from the storage server. *Sequential scans* with large I/O sizes are used in analytical workloads, such as the table scan operator in OLAP databases. In a disaggregated OLAP setup, the compute node often sequentially fetches many pages, with performance primarily determined by throughput rather than the latency of individual I/O operations.

Since we focus on I/O performance, we accessed the data from remote SSDs without processing it. However, we measured host and target CPU utilization to ensure sufficient CPU cycles remain available for computation in disaggregated query processing.

Setup. We used two nodes (one host and one target) running Ubuntu 22.04.3 LTS with the Linux 6.6.1 kernel. Each node is equipped with two Intel Xeon Gold 5220 CPUs (18 cores each), 512 GB of RAM split across both sockets, and four consumer-grade Samsung 980 Pro 1TB SSDs connected through an ASRock Hyper Quad M.2 PCIe card. The nodes are connected via an InfiniBand network using Mellanox ConnectX-5 MT27800 NICs (InfiniBand EDR 4x, 100 Gbps). We configured NVMe-oF using the MLNX_OFED drivers in version 5.9-0.5.6 on both nodes. The SSD read and write specification numbers [Sa21], as well as the NIC bandwidth, are annotated in the following figures.

Microbenchmarks were conducted using `fio` (version 3.37) [Ax24] and SPDK (version v24.05) [ST24b] to evaluate the two NVMe-oF stacks. While it is possible to mix stacks, such as using the SPDK host with the kernel target, we leave this exploration for future work. With the NVMe-oF kernel stack, two asynchronous I/O interfaces are primarily benchmarked: io_uring [LKT19], a modern and efficient interface introduced in kernel 5.1, and libaio, the traditional one that has been available since kernel 2.6. In libaio, completed I/O events are notified via hardware interrupts. In contrast, io_uring allows disabling interrupts and polling from the completion queue (CQ polling), which is the method chosen for this section. In this work, we refer to I/O depth as the number of outstanding I/O requests queued simultaneously in the I/O interface. As a tunable parameter, a high I/O depth stresses bandwidth but increases latency, while a low one minimizes latency but may underutilize bandwidth. Additionally, as a tunable parameter, the number of threads refers to host-side threads that issue I/O requests to storage. For optimal performance, we disabled most operating system features such as the file system, RAID, and kernel page cache, following recommendations from [HHL20].

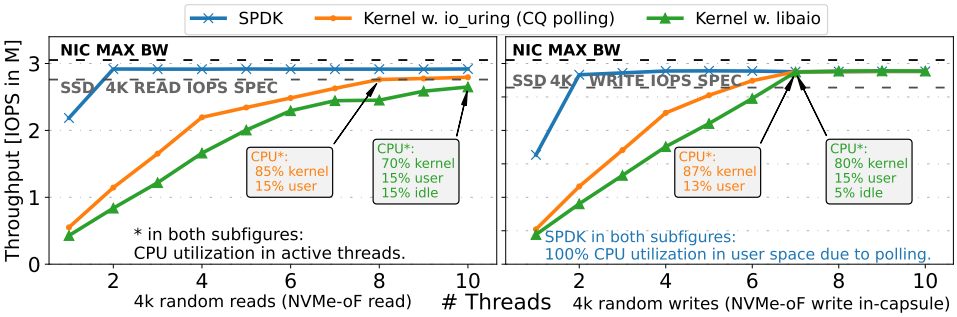

Fig. 3: 4K random I/O throughput with varying numbers of threads (I/O depth per SSD fixed at 128).

3.2 [Host-Side] Thread Impact of I/O Interfaces

In the first experiment, we evaluated random reads and writes across using a fixed page size of 4KB, which is typical in disk-based OLTP workloads. We varied thread counts on the host to examine the impact of NVMe-oF host threads on the performance. Fig. 3 illustrates that with both libaio and io_uring, network bandwidth is not fully saturated for random reads, but can be achieved for random writes. The 4K random write performance is better than the read for two main reasons: first, our SSDs perform slightly better on writes; second, with 4K NVMe-oF write in-capsule, the data is sent along with the I/O command, eliminating the need for an additional RDMA_READ call for each 4K write. With 10 threads in kernel I/O interfaces, most time and cycles are spent inside the kernel as annotated in Fig. 3. In contrast, SPDK delivers significantly better performance, achieving full bandwidth with only two threads on the NVMe-oF host.

Findings: In a disaggregated database, the compute node acts as an NVMe-oF host to access remote storage. We find that the SPDK host is preferable to the kernel host for compute nodes, as it is efficient in CPU usage and delivers high-performance I/O without any specialized configurations. With the SPDK NVMe-oF host at the compute node, it remains possible to use both SPDK and kernel I/O interfaces simultaneously, as they operate independently without interference: the SPDK NVMe-oF host manages remote storage, while the kernel I/O stack could still handle local storage, and the kernel NVMe-oF host stack for other remote storage. This is a significant difference compared to using SPDK for direct-attached NVMe SSDs, where the kernel loses access to the SSDs because SPDK requires root privileges and exclusive control of the entire drive.

3.3 [Target-Side] CPU Utilization

An interesting aspect is to analyze the CPU on the target (i.e., storage server) side. We measured CPU utilization for the different NVMe-oF target stacks (see Tab. 1), at the peak

NVMe-oF Target Stack	CPU Utilization 4k Random Read Workload	Additional Remark
SPDK	**500%**	Core count configurable Minimum 5 pinned cores for optimal performance
Kernel	**565%**	Core count non-configurable 8 active cores, each running at 70% load
Target offload	**2%**	Target offload reduces CPU utilization efficiently. However, I/O performance is suboptimal

Tab. 1: CPU utilization of the NVMe-oF target stacks at their respective maximum throughput.

throughput achieved in the previous random read workload. With SPDK, only five pinned cores at the target were sufficient to achieve the best performance. In contrast, using the kernel stack at the target caused a total CPU utilization of 565%, spread over eight cores. Unlike SPDK, the kernel drivers do not allow the I/O cores to be pinned. Target offloading can be used to lower CPU utilization at the target when using the kernel stack. We analyzed this feature. However, the obtained throughput is limited to only 1.5 GB/s at the host side. This issue in the ConnectX-5 NIC was also observed in previous work [Ch23]. Recent research [Xu24] suggests that target offloading has been improved on recent DPUs which we aim to analyze in the future.

Findings: In a real-world disaggregated setup, where storage servers typically have weak CPUs providing storage devices, the SPDK target is highly attractive to be used at storage servers. The reason is that the SPDK user-space design allows efficient CPU utilization by pinning cores and busy polling. As discussed before, the SPDK target requires unbinding NVMe devices from native kernel drivers and re-binding them to the SPDK driver for full control. However, we think this is not a limitation for many disaggregated scenarios, since multiple compute nodes across the network can still share the device.

3.4 [Host-Side] Latency at Varying Throughput Levels

Latency is crucial, especially for OLTP workloads. To examine NVMe-oF I/O latency, we measured the best latencies we can achieve on a broad range of throughput levels. The results are shown in Fig. 4. For each I/O interface, the lowest latency at each throughput level was determined by comparing the results from various I/O depth and thread configurations (provided as annotations in Fig. 4 and omitted if identical to the left). As we can see, the SPDK stack on the host requires only two host threads to reach the best latency at all throughput levels. The efficiency of SPDK is evident, as it consistently delivers the lowest latency for achieving the desired throughput, and is the only one capable of reaching 3M IOPS which fully saturates the network. Interestingly, the io_uring polling (kernel) interface also performs efficiently, though it consistently shows slightly higher latency than SPDK. Also, typically, it requires more threads and/or higher I/O depths. In contrast, the libaio (kernel) interface has the highest latency due to its inefficiencies and interrupt-based design.

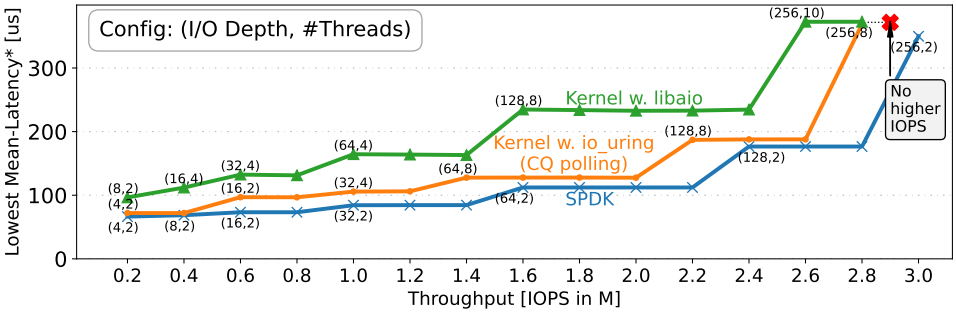

Fig. 4: Latency v.s. throughput for 4K random reads.

Findings: For latency-sensitive workloads in disaggregated database storage using NVMe-oF, relying on I/O polling interfaces such as io_uring or SPDK is advisable. The latency levels of both are primarily determined by I/O depth. The latency differences between the two polling I/O interfaces are relatively small. However, the NVMe-oF kernel stack is more complex than a general-purpose I/O interface, as it also includes drivers specific to the NVMe-oF protocol. This raises concerns about potential issues arising from interactions between these components within the kernel stack, as discussed next.

3.5 [Kernel-Stack] NVMe-oF and Local Comparison

We conducted the following experiment to understand the differences between NVMe-oF and local NVMe. With the kernel stack, a key difference between NVMe-oF and direct-attached block devices is the submission queue size `sqsize` of the NVMe controller. For direct-attached NVMe SSDs, `sqsize` is generally determined by the device hardware capabilities. Surprisingly, however, in NVMe-oF kernel setups, `sqsize` is set to a low constant value by the NVMe-oF host and target driver. In our setup, the `sqsize` was only 128 for the NVMe-oF host, whereas for direct-attached SSD it was 1024.

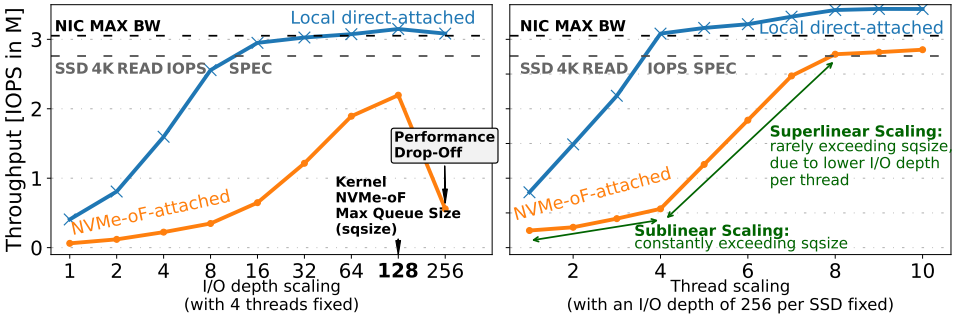

Fig. 5: 4K random read performance of local and NVMe-oF SSDs using io_uring (CQ polling).

To illustrate how differences in `sqsize` affect I/O interface performance, we evaluated the same workload on both direct-attached and NVMe-oF SSDs. While io_uring demonstrates efficiency with local SSDs, a significant throughput drop is observed with NVMe-oF as the I/O depth increases to 256, as shown on the left side of Fig. 5. This performance issue arises because the number of outstanding I/O requests in I/O interfaces exceeds `sqsize` in the NVMe controller, causing io_uring threads to poll and re-submit excess requests until they succeed continuously. Moreover, it adds pressure on system resources (PSI and Memcg) as more threads compete for accounting resources. Focusing on an I/O depth of 256 per SSD, the right side of Fig. 5 reveals an unusual pattern: sublinear scaling up to 4 threads, followed by superlinear scaling up to around 8 threads. The superlinear scaling occurs because, with more threads per SSD, each thread issues fewer I/O requests, reducing the over-driving frequency.

Findings: When designing a disaggregated storage engine for high random I/O performance, the kernel stack can not fully exploit the capabilities of modern storage devices. First, compared to direct-attached configurations, simply increasing the number of outstanding NVMe-oF I/O requests does not always compensate for the network latency introduced. The low queue size limit (128 in our setup) in the NVMe controller can easily be exceeded in I/O interfaces, possibly leading to a significant performance drop, as illustrated in Fig. 5. Second, fully saturating modern storage devices is neither easy nor trivial, as each device with PCIe 4.0 or higher typically requires more than 300 outstanding I/O requests to achieve full utilization [HL23]. In contrast, SPDK offers greater setup flexibility due to its user-space nature, enabling easy configuration of equivalent parameters such as NVMe controller queue size.

3.6 [Host-Side] Large-Sized Sequential I/O Performance

Large sequential reads are common in OLAP workloads. We evaluated such a workload as illustrated in Fig. 6. Interestingly, using an I/O depth of 16 with a single thread provides good results across the different stacks, as we can see. For example, SPDK requires a minimum I/O size of 16 KB to saturate the network, while io_uring and libaio need only 32 KB and 64 KB, respectively, which are not uncommon for OLAP. As a result, one thread of the host side is sufficient to fully saturate network bandwidth across these I/O interfaces. An exception is the POSIX `pread` I/O interface, which inherently operates with an I/O depth of 1 due to its blocking nature. Even with 16 threads, it fails to saturate the network, which is the primary bandwidth limitation in this setup.

Findings: For designing a disaggregated OLAP database with NVMe-oF, the io_uring polling interface is sufficient to fully utilize NIC capabilities performing large sequential read access patterns. Once network bandwidth is saturated, io_uring matches SPDK in both I/O bandwidth and CPU overheads, indicating that polled io_uring does not introduce significant overheads within the kernel during large sequential reads. Therefore, for disaggregated OLAP databases using NVMe-oF, it is recommended to deploy the kernel NVMe-oF stack as a native solution, as it does not bottleneck performance.

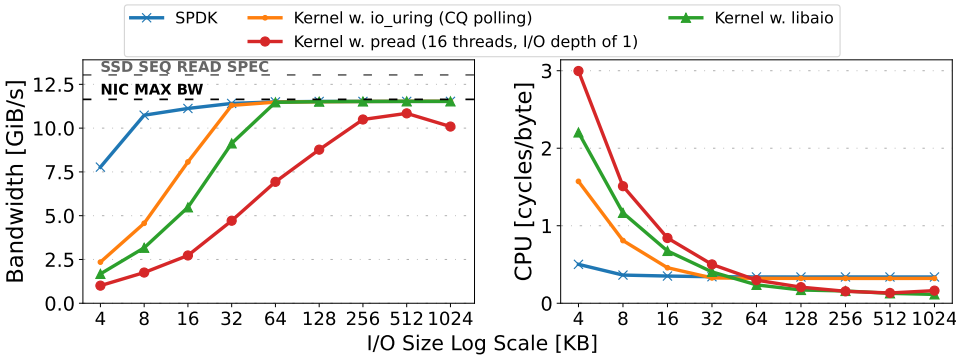

Fig. 6: Impact of the I/O size on sequential reads (with one thread and an I/O depth of 16 fixed).

4 Conclusions and Future Directions

In this paper, we explored how to best leverage NVMe-oF for database system workloads. We summarize our findings as follows:

At the compute node, acting as the NVMe-oF host, SPDK consistently outperforms kernel I/O interfaces across all workloads by offering high I/O performance and low CPU overhead. Moreover, the SPDK host at the compute node does not require device driver unbinding. However, as a specialized I/O stack, SPDK has usability limitations – keeping I/O results and statistics in user-space, invisible to the kernel. Additionally, it is challenging to rewrite existing compute node applications to adopt the SPDK I/O stack.

At the storage node, acting as the NVMe-oF target, with weak CPUs provisioning large storage pools, there is a trade-off in selecting the target stack. Importantly, the kernel target lacks advanced control options, such as core pinning or configuring the NVMe controller queue size. As such, the SPDK target provides better CPU efficiency through polling and core pinning. However, SPDK's major drawbacks include requiring root privileges and exclusive access to the entire drive, making it less practical in the cloud.

Performance differences between NVMe-oF and local storage are not solely due to the added latency of the network but it also stems from inefficiencies within the NVMe-oF kernel stack such as settings regarding the NVMe controller queue size. In contrast, SPDK provides again greater flexibility in configuration without kernel constraints.

Overall, we think NVMe-oF is an underexplored protocol for disaggregated databases. In the future, we thus aim to explore storage engine designs such as [ZBL22] to make use of NVMe-oF.

Acknowledgments

We gratefully acknowledge the support by the Federal Ministry of Education and Research (BMBF) under Grant No. 01IS22091. We also thank hessian.AI and DFKI for their support.

Bibliography

[Ax24] Axboe, Jens: fio - Flexible I/O Tester, 2024. https://github.com/axboe/fio.

[Ba23] Bai, Wei; Abdeen, Shanim Sainul; Agrawal, Ankit; Attre, Krishan Kumar; Bahl, Paramvir; Bhagat, Ameya; Bhaskara, Gowri; Brokhman, Tanya; Cao, Lei; Cheema, Ahmad; Chow, Rebecca; Cohen, Jeff; Elhaddad, Mahmoud; Ette, Vivek; Figlin, Igal; Firestone, Daniel; George, Mathew; German, Ilya; Ghai, Lakhmeet; Green, Eric; Greenberg, Albert G.; Gupta, Manish; Haagens, Randy; Hendel, Matthew; Howlader, Ridwan; John, Neetha; Johnstone, Julia; Jolly, Tom; Kramer, Greg; Kruse, David; Kumar, Ankit; Lan, Erica; Lee, Ivan; Levy, Avi; Lipshteyn, Marina; Liu, Xin; Liu, Chen; Lu, Guohan; Lu, Yuemin; Lu, Xiakun; Makhervaks, Vadim; Malashanka, Ulad; Maltz, David A.; Marinos, Ilias; Mehta, Rohan; Murthi, Sharda; Namdhari, Anup; Ogus, Aaron; Padhye, Jitendra; Pandya, Madhav; Phillips, Douglas; Power, Adrian; Puri, Suraj; Raindel, Shachar; Rhee, Jordan; Russo, Anthony; Sah, Maneesh; Sheriff, Ali; Sparacino, Chris; Srivastava, Ashutosh; Sun, Weixiang; Swanson, Nick; Tian, Fuhou; Tomczyk, Lukasz; Vadlamuri, Vamsi; Wolman, Alec; Xie, Ying; Yom, Joyce; Yuan, Lihua; Zhang, Yanzhao; Zill, Brian: Empowering Azure Storage with RDMA. In: NSDI. USENIX Association, pp. 49–67, 2023.

[Bi16] Binnig, Carsten; Crotty, Andrew; Galakatos, Alex; Kraska, Tim; Zamanian, Erfan: The End of Slow Networks: It's Time for a Redesign. Proc. VLDB Endow., 9(7):528–539, 2016.

[BJZ24] Binder, Simon; Jasny, Matthias; Ziegler, Tobias: Seamless: Transparent Storage Access Through Smart Switches. In (Binnig, Carsten; Tatbul, Nesime, eds): Proceedings of the 20th International Workshop on Data Management on New Hardware, DaMoN 2024, Santiago, Chile, 10 June 2024. ACM, pp. 13:1–13:5, 2024.

[Ch23] Chen, Yiquan; Chen, Jinlong; Wang, Yijing; Chen, Yi; Jin, Zhen; Xu, Jiexiong; Fang, Guoju; Lin, Wenhai; Wei, Chengkun; Chen, Wenzhi: HyQ: Hybrid I/O Queue Architecture for NVMe over Fabrics to Enable High- Performance Hardware Offloading. In: 23rd IEEE/ACM International Symposium on Cluster, Cloud and Internet Computing, CCGrid 2023, Bangalore, India, May 1-4, 2023. IEEE, pp. 13–24, 2023.

[DC20] Davis, Rob; Cebeli, Ilker: Accelerating NVMe over Fabrics with Hardware Offloads at 100Gb/s and Beyond, 2020. https://nvmexpress.org/wp-content/uploads/Accelerating-NVMe-over-Fabrics-with-Hardware-Offloads.pdf.

[Dr14] Dragojevic, Aleksandar; Narayanan, Dushyanth; Castro, Miguel; Hodson, Orion: FaRM: Fast Remote Memory. In (Mahajan, Ratul; Stoica, Ion, eds): Proceedings of the 11th USENIX Symposium on Networked Systems Design and Implementation, NSDI 2014, Seattle, WA, USA, April 2-4, 2014. USENIX Association, pp. 401–414, 2014.

[Dr15] Dragojevic, Aleksandar; Narayanan, Dushyanth; Nightingale, Edmund B.; Renzelmann, Matthew; Shamis, Alex; Badam, Anirudh; Castro, Miguel: No compromises: distributed transactions with consistency, availability, and performance. In (Miller, Ethan L.; Hand, Steven, eds): Proceedings of the 25th Symposium on Operating Systems Principles, SOSP 2015, Monterey, CA, USA, October 4-7, 2015. ACM, pp. 54–70, 2015.

[Ex21] NVMe over Fabrics (oF) Specification (historical reference only), https://nvmexpress.org/specification/nvme-of-specification/.

[Ex24] NVM Express specification, http://www.nvmexpress.org/specifications/.

[Gu18] Guz, Zvika; Li, Harry (Huan); Shayesteh, Anahita; Balakrishnan, Vijay: Performance Characterization of NVMe-over-Fabrics Storage Disaggregation. ACM Trans. Storage, 14(4):31:1–31:18, 2018.

[HHL20] Haas, Gabriel; Haubenschild, Michael; Leis, Viktor: Exploiting Directly-Attached NVMe Arrays in DBMS. In: 10th Conference on Innovative Data Systems Research, CIDR 2020, Amsterdam, The Netherlands, January 12-15, 2020, Online Proceedings. www.cidrdb.org, 2020.

[HL23] Haas, Gabriel; Leis, Viktor: What Modern NVMe Storage Can Do, And How To Exploit It: High-Performance I/O for High-Performance Storage Engines. Proc. VLDB Endow., 16(9):2090–2102, 2023.

[Li16] Li, Feng; Das, Sudipto; Syamala, Manoj; Narasayya, Vivek R.: Accelerating Relational Databases by Leveraging Remote Memory and RDMA. In: SIGMOD Conference. ACM, pp. 355–370, 2016.

[LKT19] Linux-Kernel-Team: Efficient IO with io_uring, 2019. https://kernel.dk/io_uring.pdf.

[NV20] NVIDIA Mellanox ConnectX-5 Product Brief, https://network.nvidia.com/files/doc-2020/pb-connectx-5-en-card.pdf.

[Sa21] Samsung V-NAND SSD 980 PRO - Data Sheet, https://download.semiconductor.samsung.com/resources/data-sheet/Samsung-NVMe-SSD-980-PRO-Data-Sheet_Rev.2.1_230509_10129505081019.pdf.

[ST24a] SPDK-Team: SPDK 24.05 NVMe-oF RDMA Performance Report (Mellanox ConnectX-5), 2024. https://ci.spdk.io/download/performance-reports/SPDK_rdma_mlx_perf_report_2405.pdf.

[ST24b] SPDK-Team: Storage Performance Development Kit (SPDK), 2024. https://spdk.io/.

[Xu24] Xu, Jiexiong; Qiu, Yue; Chen, Yiquan; Wang, Yijing; Lin, Wenhai; Lin, Yiquan; Zhao, Shushu; Liu, Yuqi; Wang, Ying; Chen, Wenzhi: Performance Characterization of SmartNIC NVMe-over-Fabrics Target Offloading. In: Proceedings of the 17th ACM International Systems and Storage Conference. SYSTOR '24, Association for Computing Machinery, New York, NY, USA, p. 14–24, 2024.

[Ya17] Yang, Ziye; Harris, James R.; Walker, Benjamin; Verkamp, Daniel; Liu, Changpeng; Chang, Cunyin; Cao, Gang; Stern, Jonathan; Verma, Vishal; Paul, Luse E.: SPDK: A Development Kit to Build High Performance Storage Applications. In: IEEE International Conference on Cloud Computing Technology and Science, CloudCom 2017, Hong Kong, December 11-14, 2017. IEEE Computer Society, pp. 154–161, 2017.

[Za17] Zamanian, Erfan; Binnig, Carsten; Harris, Tim; Kraska, Tim: The end of a myth: distributed transactions can scale. Proc. VLDB Endow., 10(6):685–696, February 2017.

[Za21] Zamanian, Erfan; Shun, Julian; Binnig, Carsten; Kraska, Tim: Chiller: Contention-centric Transaction Execution and Data Partitioning for Modern Networks. SIGMOD Rec., 50(1):15–22, 2021.

[ZBL22] Ziegler, Tobias; Binnig, Carsten; Leis, Viktor: ScaleStore: A Fast and Cost-Efficient Storage Engine using DRAM, NVMe, and RDMA. In (Ives, Zachary G.; Bonifati, Angela; Abbadi, Amr El, eds): SIGMOD '22: International Conference on Management of Data, Philadelphia, PA, USA, June 12 - 17, 2022. ACM, pp. 685–699, 2022.

[Zi19] Ziegler, Tobias; Vani, Sumukha Tumkur; Binnig, Carsten; Fonseca, Rodrigo; Kraska, Tim: Designing Distributed Tree-based Index Structures for Fast RDMA-capable Networks. In (Boncz, Peter A.; Manegold, Stefan; Ailamaki, Anastasia; Deshpande, Amol; Kraska, Tim, eds): Proceedings of the 2019 International Conference on Management of Data, SIGMOD Conference 2019, Amsterdam, The Netherlands, June 30 - July 5, 2019. ACM, pp. 741–758, 2019.