Seamless: Transparent Storage Access Through Smart Switches

Simon Binder
TU Darmstadt

Matthias Jasny
TU Darmstadt

Tobias Ziegler
TU Darmstadt

ABSTRACT

This paper presents Seamless, a switch-based accelerator for disaggregated SSD-based systems. Seamless comprises two fundamental components: (1) a hardware-accelerated, unified remote storage protocol that ensures efficient data access to Flash and remote memory. (2) A hardware-accelerated concurrency protocol with a simple interface like a buffer manager: `fix(page_id, {exclusive, shared})` for acquiring page latches and `unfix(page_id)` for releasing them. Seamless achieves this by using a P4-programmable switch. To enable transparent storage access and caching, we maintain each page's location within the switch, whether in memory or on an SSD. Upon receiving a page request, Seamless determines the page's location and rewrites the communication protocol at line rate to NVMe-oF or RDMA accordingly. We manage the latches directly on the switch to maintain consistency and provide concurrency control. We show initial promising results for switch-based acceleration: Seamless hard-level accelerated protocol outperforms a software-based RDMA-optimized system by 20%.

CCS CONCEPTS

• **Information systems → Parallel and distributed DBMSs**; • **Networks → In-network processing**.

ACM Reference Format:
Simon Binder, Matthias Jasny, and Tobias Ziegler. 2024. Seamless: Transparent Storage Access Through Smart Switches. In *20th International Workshop on Data Management on New Hardware (DaMoN '24), June 10, 2024, Santiago, AA, Chile.* ACM, New York, NY, USA, 5 pages. https://doi.org/10.1145/3662010.3663446

1 INTRODUCTION

RDMA & Disaggregation. Many modern cloud databases adopt a disaggregated architecture to independently scale storage and compute resources. Given the central role of networking in such architectures, it is unsurprising that low-latency Remote Direct Memory Access (RDMA) has become an indispensable tool [1, 10]. By leveraging RDMA, systems can directly access data on remote memory within single-digit microseconds.

Economic Shift: DRAM vs. Flash. So far, RDMA-enabled systems have predominantly been fully in-memory. While these systems are very fast, memory prices have started to plateau, reducing their economic feasibility. At the same time, flash s torage c osts have decreased by 30x over the past decade, making Flash an attractive

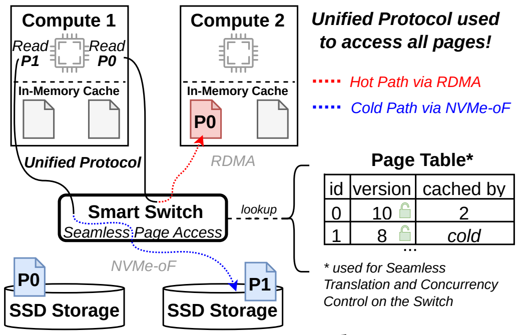

Figure 1: System overview. SSDs and remote memory are exposed under a unifying protocol

storage medium. Consequently, there is a shift from in-memory to flash-based storage engines [5, 9, 12, 16].

RDMA and Flash: Not a Good Fit. Disaggregated systems must integrate flash-based storage with RDMA-based data access to achieve both speed and cost-efficiency. Such an integration, however, presents several challenges: Firstly, RDMA was inherently designed for memory-centric operations, not for interfacing directly with flash storage. Secondly, although new protocols like NVMe over Fabrics (NVMe-oF) allow direct access to flash over fabric, they differ significantly from traditional RDMA operations. Consequently, a database must distinguish between remote SSD and memory access to support both, adding complexity.

Caching and Consistency: Integration made complex. The complexity is further compounded by the need for caching to bridge the gap in access latency. Flash storage has an access latency of 70 μs, an order of magnitude higher than RDMA's latency [14]. We could use dynamic caching to hide this latency gap, in which hot data resides in the aggregated memory of all compute nodes, while Flash is used for colder data. However, such caching approaches require coherence protocols to provide data consistency under concurrent updates. While software-based solutions like ScaleStore [17] attempt to address this with complex inter-node communication protocols, these protocols require multiple round-trips, introducing overhead.

Seamless: Bridging RDMA and NVMeOF with Smart Switches. This paper presents Seamless, a switch-based accelerator. Seamless consists of two fundamental components: (1) a hardware-accelerated, transparent remote storage protocol that ensures efficient data access to Flash and remote memory. (2) a hardware-accelerated concurrency protocol with a simple interface like a buffer manager: `fix(page_id, {exclusive, shared})` for acquiring page latches and `unfix(page_id)` for releasing them. Seamless achieves this by using a P4-programmable switch as described below.

2 SYSTEM OVERVIEW

In this section, we illustrate the main concepts behind our system.

2.1 Seamless Transparent Access and Caching

To implement caching, software-based protocols require additional round-trips to resolve the page's location and state before reading it. In contrast, our protocol aims to provide coherent data access without additional round-trips.

Transparent Access through Protocol Rewrite. To simplify, consider a static page allocation first: Cold pages reside on SSDs, and hot pages are cached in remote memory. For instance, $P0$ on compute 2 and $P1$ on SSD in Figure 1. Normally, we need two protocols: NVMe-oF for SSD access and direct RDMA primitives for retrieving data from remote memory. However, Seamless hides these details with a uniform protocol. The core concept of our protocol is the transparent rewriting of read requests at line rate. Under our protocol, a compute node requests a page using an RDMA `READ` operation, specifying a logical page ID instead of a remote memory location. This RDMA request is intercepted by the switch where we use on-switch memory to store a mapping from page ID to the page location (Page Table in Figure 1). This enables our data plane to perform semantic routing and transparently rewrite the protocol to NVMe-oF or replace the page ID with the current remote memory location. The requesting node will eventually receive the page without knowing whether it came from SSD or remote memory.

NVMe-oF protocol integration. To copy data from remote memory into a local buffer, nodes typically use a one-sided RDMA `READ` operation specifying the remote address as well as a length (cf. Figure 2 (left)). This operation is handled on the NIC of the receiving node without involving its CPU, enabling memory access with minimal overhead. Compared to raw RDMA, the exchange for NVMe-oF reads involves more packets (see Figure 2 (right)): To read data from SSDs into a local buffer, nodes post an RDMA `SEND` message towards the SSD controller indicating which pages to read as well as a destination buffer. Once the pages have been read from flash memory, the SSD controller issues an RDMA `WRITE` to store their contents in the designated target buffer. Finally, it issues another `SEND` to report the operation as completed.

Unifying these access patterns was a challenge in Seamless's design. To maintain the benefit of remote CPU-bypass of one-sided RDMA and avoid the costs of `SEND` operations [18], our initial experiments used a data plane transparently rewriting `WRITE`s from the NVMe controller into `READ` responses, allowing nodes to issue RDMA reads to any page, regardless of whether it exists in memory. However, the high access latencies for SSDs did not align with the assumptions of the one-sided RDMA protocol, causing excessive retransmits and out-of-order responses for mixed remote memory and SSD reads. This results in severe performance and reliability issues, making this approach unviable.

To maintain compatibility with the NVMe communication scheme, our uniform protocol must also decouple read requests and data transfer. Seamless read requests use RDMA `READ`s encoding the target page address and an index into the local buffer pool in the RDMA destination address. The response for this read is a status code containing the generated NVMe command id or an error code if the request was aborted due to, e.g., the page being locked. For reads routed to SSDs, our data plane takes the role of the host in the NVMe scheme as shown in Figure 2 (right) by generating a new command ID and sending an NVMe-oF command to the SSD controller. Writes issued by the controller are forwarded to the destination on the compute node and the final NVMe completion message is translated into a uniform format, allowing compute nodes to recognize completions after the transfer. By implementing the NVMe-oF protocol in our data plane, Seamless is compatible with existing NVMe-oF deployments instead of requiring special hardware on SSD nodes.

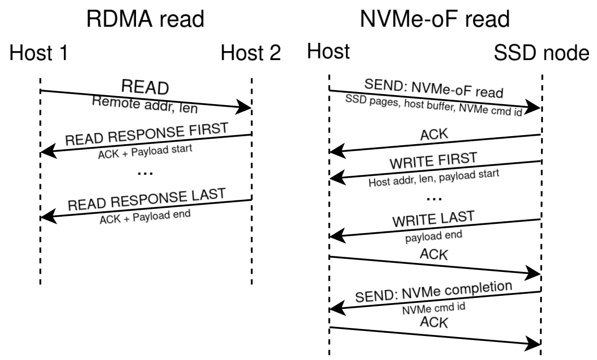

Figure 2: Packets exchanged for RDMA reads (left) and NVMe-oF reads using RDMA (right). The initiating node for data transfers is inverted for NVMe-oF, complicating a direct translation between the protocols.

Dynamic Caching. While we discussed a static page allocation, actual access patterns are dynamic, requiring to evict or maintain pages in the in-memory caches of compute nodes. To support this requirement and perform on-the-fly routing decisions between SSD and memory accesses, we introduce a 'cached-by' field in the switch's table, indicating the node that caches each page as illustrated in Figure 1. Further, to fully leverage our transparent access protocol, we update the switch's page table to reflect the changing access patterns, i.e., when a page is evicted to SSD or cached in a new remote node. By piggybacking this update mechanism with the NVMe-oF protocol we avoid extra network traffic for the update process. When a node requests a page currently stored on SSD, the switch rewrites this request as an NVMe-oF request. As the SSD completes the request, the switch observes a NVMe-oF completion packet and updates the 'cached-by' field. When a node evicts a page with modifications, a write request is sent to the switch which translates it into an NVME-oF write command towards the correct SSD. Direct access to the SSDs from nodes is not possible as the connection state has been offloaded to the switch. Moreover, nodes reject further reads to evicted pages, causing reads to be forwarded to SSDs which triggers the automatic update of the 'cached-by' field as described above.

2.2 Seamless Concurrency Control

Besides providing transparent data access and caching, Seamless can simplify and optimize concurrent access control to pages. Many software-based solutions implement relatively complex MESI-like protocols [4, 17] to ensure coherence and concurrency. Our protocol is simpler and leverages the on-switch memory and the switch's low latency (the switch is in the middle of normal communication between nodes, the latency to the switch and back is 1/2 round-trip time). Seamless stores a lock bit alongside the caching and location details in the switch's page table. Thus, when a client requests a page, the switch can efficiently verify the lock status.

Optimistic Concurrency Control. We use optimistic concurrency control to mitigate the overhead associated with distributed lock protocols and alleviate RDMA synchronization bottlenecks. For that, we add a 'latch' field in the switch's page table, essentially a 64-bit field comprising version and lock-bit [3].

For shared access, the data plane examines the lock bit; if it indicates exclusive locking, the request is denied. Then, nodes will continue to retry until the page latch is available. If the lock bit is unset, the switch provides the current version number and the page. When the client finishes its task on the page, the version number is reverified to detect concurrent writes.

Obtaining an exclusive page reference requires atomically setting the lock bit, which is performed on the switch. This leverages the switch's linear execution model, which ensures packets are processed sequentially at line rate. Releasing the exclusive reference is similarly straightforward. It involves incrementing the version number and reseting the lock bit.

Linear Execution Ensures Correctness. Implementations of optimistic synchronization protocols over RDMA often face correctness issues. Specifically, ensuring consistency within a single RDMA READ request is hard (cf. [19] for more details). Although some leverage the RDMA card's memory read ordering, these mechanisms lack formal specification. The advantage of processing RDMA on a switch is its guarantee of linear packet processing, enabling the following optimization:

When a reader requests a page from SSD, the switch provides the version number and converts the request into a NVMe-oF operation. Typically, NVMe-oF reads decompose into multiple data packets. As these packets are being processed, another node could attempt to modify the SSD page. This modification requires an exclusive lock and a version update, which the switch allows under our optimistic protocol (readers do not lock). At the same time, the switch alters NVMe-oF completion packets sent by SSDs after a completed read to include the latest version number in its response. The updated version number effectively indicates the potential page modification to the client, as described earlier. This mechanism, unique to our integrated switch model, ensures consistent page reads with a single invocation, which is typically hard to achieve with RDMA only.

Software-based fallback. Due to being restricted to a few MB of on-switch memory, storing the full page table of entire workloads on the switch is unfeasible. Thus, we only index the most frequently accessed pages in the on-switch page table. State for remaining pages is stored on a memory server. For these pages, no inline translation in the network path is possible. Instead, nodes rely on accessing that page table explicitly via RDMA to determine how pages need to be accessed. Having a separate fallback mechanism also serves as a baseline to evaluate the switch-based optimizations against.

3 EVALUATION

Index lookups. To show Seamless' efficiency improvements over software-based systems, we compare it with Scalestore, a distributed storage engine with an RDMA-optimized coherence protocol. In

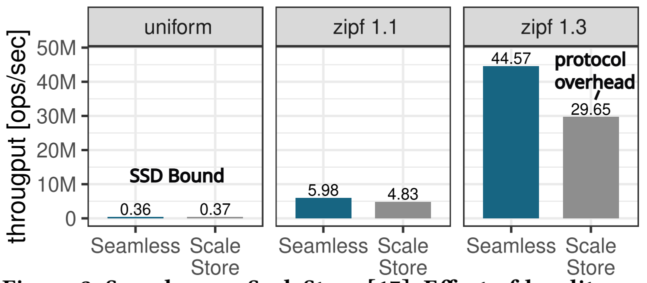

Figure 3: Seamless vs. ScaleStore [17]: Effect of locality on lookup performance on a 128 GiB **data set (2 compute nodes each with** 64 GiB **buffer, 1 storage node)**

our experiment, we deploy a single storage node equipped with four 1 TB SSDs, and two compute nodes with 28 threads and a 64 GiB local page buffer. We are using 64 KiB as the page size to optimize NVMe-oF throughput [8]. The queue depths of NVMe and RDMA are 128. To simulate typical workload scenarios, we implemented a read-only micro-benchmark similar to YCSB workload C. We examined the system throughput on a data set of 128 GiB with varying data access localities, ranging from uniform to a Zipfian distribution, with a skew of 1.3. This allows us to show how a varying number of local, cache, and SSD accesses impacts system performance. As shown in Figure 3, a uniform distribution is the most challenging workload since most accesses result in SSD reads. Both Seamless and ScaleStore experience many page faults and thus show similar performance (both are IO-bound on our SSDs). When we increase the locality ($z = 1.1$), the performance increases for both systems since there are more local and remote cache hits. However, the Seamless protocol is 20% more efficient than ScaleStore's software-based protocol. The reason for this is two-fold: 1) Our P4 switch implementation rewrites the protocol at line rate, and 2) since the switch is on the communication path, this does not add additional overhead. Note that, regardless of access skew, Seamless is guaranteed to run at line rate due to the P4 compiler's strict real-time checking, i.e., if the program is not runnable at line rate, it will not compile [2]. With even more locality ($z = 1.3$), the efficiency improvements become more pronounced, and Seamless's unified protocol, combined with our caching and latching strategy, consistently outperforms ScaleStore's coherence protocol.

Latency breakdown. To verify the impact of transparent reads implemented on the switch, we ran microbenchmarks exercising only the network layer, avoiding distortions from other parts of the system, like the local page buffer. We measure latencies in three setups simulating different sources for page reads: 1) Cold reads from the SSD, 2) Hot reads from remote memory, and 3) local cache hits, which avoid data transfer but still require a round-trip to verify the lock state. Measurements were made on a single node with 28 threads each using an RDMA queue with a depth of 128. A second passive compute node served as a target for remote-memory reads. For this experiment, we increase the target throughput until the system can no longer sustain the required throughput. Based on the target throughput, every worker is assigned a schedule when it needs to send the next operation. This schedule is generated based on a Poisson process, where the time between two operations is exponentially distributed. When a worker misses a scheduled operation, we send it as soon as possible and include the wait time, i.e.,

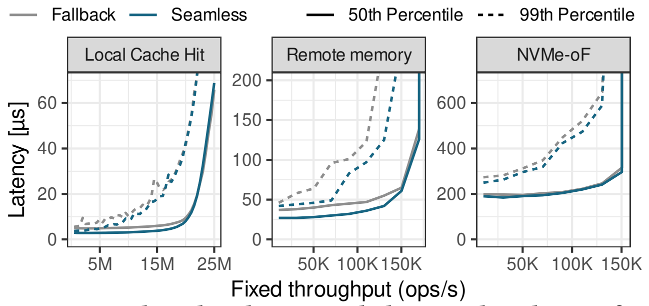

Figure 4: 50th and 99th percentile latency distribution for Seamless operations at Poisson-distributed rates. Seamless is most impactful for hot data being read from remote memory and for local cache hits which only require a round-trip to verify the lock bit.

we correct the latencies. This means latencies spike when the target throughput is no longer sustainable. Recall that, with the optimizations enabled, all three read kinds are indistinguishable from the perspective of a compute node. Without the switch-enabled transparent access optimization, lookups require an additional round-trip to load the current location via a page table from a remote memory node as described in Section 2.2. As Figure 4 shows, the transparent rewrite with an on-switch page table reduces latency by avoiding additional round-trips. For SSD reads, this impact is dwarfed by the intrinsically high access latencies for NVMe-oF. The impact of eliminating a full round-trip is most pronounced for remote-memory reads: Until reaching throughput rates saturating the network link, the additional round-trip needed without network optimizations increases latency. Local cache hits without data transfer still benefit from the optimization, as responses to lock-requests are created on the switch instead of requiring a full round-trip to a remote RDMA NIC.

Throughput impact of transparent routing. The restricted on-device memory necessitates using a software-based fallback requiring additional round-trips over Seamless' transparent routing optimization. To evaluate the throughput impact of our in-network optimizations, we ran a benchmark with a single node requesting pages from remote memory with and without our switch, i.e., implementing the protocol and using a remote memory node as page table. As Figure 5 shows, our optimizations increase the maximum throughput by around 12%. The restricted memory on the switch we used provides capacity for around 280 k pages, or around 17 GiB of indexed content with a page size of 64 KiB. As workloads exceed this size, reads are implemented with the software-based fallback as described in Section 2.2. The throughput rate of these workloads degrades gracefully, eventually reaching unoptimized levels for large data sets. This means that Seamless is most effective for a small working set (below 17 GiB) or skewed access patterns, such as a tree index, with frequent hits to a subset of data. Newer switches have larger memory, meaning that the restrictions are less severe for more recent hardware.

Limitations. For SSD-bound workloads with few cache hits, the uniform translation has no impact on throughput as the performance of SSDs is the limiting factor. Similarly, we observed no speedup for write-heavy workloads, as the contention caused by exclusive page pins dominates the performance characteristics of the overal system. Our results show that Seamless delivers a promising speedup for read-heavy workloads spanning across memory and SSDs.

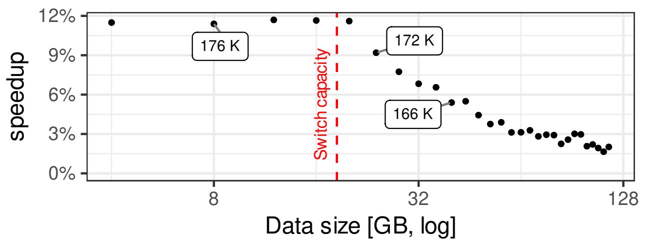

Figure 5: Relative speedup for uniform 64 KiB page reads from remote memory when enabling transparent routing. For workloads exceeding on-switch capacities, performance degrades gracefully to reach non-optimized levels.

4 RELATED WORK

Different approaches of integrating NVMe storage into distributed database systems have been explored with software-based works such as LeanStore [9], Umbra [12] or ScaleStore [17]. These contributions differ from our work in that they are not exploiting programmability in the network layer, thus exhibiting additional complexity in remote page accesses or eviction.

A technologically similar contribution from network research is SwitchKV [11], which uses a programmable switch to accelerate distributed key-value stores by redistributing hot keys across different nodes to avoid contention. While SwitchKV implements content-based routing similar to Seamless, it does not provide a uniform translation layer between different protocols.

NetCache [7] and P4DB [6] accelerate key-value stores and database systems by offloading hot tuples onto a programmable switch. Seamless only stores metadata on the switch to enable accelerated page accesses. Further, it operates on a page-level granularity instead of on individual tuples and provides an interface similar to traditional buffer managers, thus potentially making it applicable to a wider range of applications.

NetLock [15] is a network-optimized centralized lock manager achieving speedups over software-based RDMA lock managers. Unlike NetLock, Seamless employs an optimistic-concurrency scheme based on latches.

RackBlox [13] uses programmable switches and SSDs to implement I/O scheduling and SSD management like wear leveling across the cluster. Unlike our work, which puts a focus on database workloads specifically, RackBlox has no support for transparently caching pages or page-level concurrency control. Similarly, Seamless is compatible with any NVMe-oF implementation and does not require programmable SSDs.

5 FUTURE ROUTE

Programmable switches have shown promising initial results for disaggregated systems. Looking ahead, we aim to build upon these outcomes by 1) investigating fault tolerance to position the switch as an opportunistic and robust accelerator, and 2) integrating it into a full disaggregated system.

REFERENCES

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

[2] Pat Bosshart, Glen Gibb, Hun-Seok Kim, George Varghese, Nick McKeown, Martin Izzard, Fernando A. Mujica, and Mark Horowitz. 2013. Forwarding metamorphosis: fast programmable match-action processing in hardware for SDN. In *ACM SIGCOMM 2013 Conference, SIGCOMM 2013, Hong Kong, August 12-16, 2013*, Dah Ming Chiu, Jia Wang, Paul Barford, and Srinivasan Seshan (Eds.). ACM, 99–110. https://doi.org/10.1145/2486001.2486011

[3] Jan Böttcher, Viktor Leis, Jana Giceva, Thomas Neumann, and Alfons Kemper. 2020. Scalable and robust latches for database systems. In *DaMoN*. ACM, 2:1–2:8.

[4] Qingchao Cai, Wentian Guo, Hao Zhang, Divyakant Agrawal, Gang Chen, Beng Chin Ooi, Kian-Lee Tan, Yong Meng Teo, and Sheng Wang. 2018. Efficient Distributed Memory Management with RDMA and Caching. *Proc. VLDB Endow.* 11, 11 (2018), 1604–1617.

[5] Gabriel Haas and Viktor Leis. 2023. 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 (2023), 2090–2102. https://doi.org/10.14778/3598581.3598584

[6] Matthias Jasny, Lasse Thostrup, Tobias Ziegler, and Carsten Binnig. 2022. P4DB - The Case for In-Network OLTP. In *Proceedings of the 2022 International Conference on Management of Data* (Philadelphia, PA, USA) *(SIGMOD '22)*. Association for Computing Machinery, New York, NY, USA, 1375–1389. https://doi.org/10.1145/3514221.3517825

[7] Xin Jin, Xiaozhou Li, Haoyu Zhang, Robert Soulé, Jeongkeun Lee, Nate Foster, Changhoon Kim, and Ion Stoica. 2017. NetCache: Balancing Key-Value Stores with Fast In-Network Caching. In *Proceedings of the 26th Symposium on Operating Systems Principles, Shanghai, China, October 28-31, 2017*. ACM, 121–136. https://doi.org/10.1145/3132747.3132764

[8] Arjun Kashyap and Xiaoyi Lu. 2022. NVMe-oAF: Towards Adaptive NVMe-oF for IO-Intensive Workloads on HPC Cloud. In *HPDC '22: The 31st International Symposium on High-Performance Parallel and Distributed Computing, Minneapolis, MN, USA, 27 June 2022 - 1 July 2022*, Jon B. Weissman, Abhishek Chandra, Ada Gavrilovska, and Devesh Tiwari (Eds.). ACM, 56–70. https://doi.org/10.1145/3502181.3531476

[9] Viktor Leis, Michael Haubenschild, Alfons Kemper, and Thomas Neumann. 2018. LeanStore: In-Memory Data Management beyond Main Memory. In *ICDE*. IEEE Computer Society, 185–196.

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

[11] Xiaozhou Li, Raghav Sethi, Michael Kaminsky, David G. Andersen, and Michael J. Freedman. 2016. Be Fast, Cheap and in Control with SwitchKV. In *13th USENIX Symposium on Networked Systems Design and Implementation, NSDI 2016, Santa Clara, CA, USA, March 16-18, 2016*, Katerina J. Argyraki and Rebecca Isaacs (Eds.). USENIX Association, 31–44. https://www.usenix.org/conference/nsdi16/technical-sessions/presentation/li-xiaozhou

[12] Thomas Neumann and Michael J. Freitag. 2020. Umbra: A Disk-Based System with In-Memory Performance. In *CIDR*. www.cidrdb.org.

[13] Benjamin Reidys, Yuqi Xue, Daixuan Li, Bharat Sukhwani, Wen-Mei Hwu, Deming Chen, Sameh W. Asaad, and Jian Huang. 2023. RackBlox: A Software-Defined Rack-Scale Storage System with Network-Storage Co-Design. In *Proceedings of the 29th Symposium on Operating Systems Principles, SOSP 2023, Koblenz, Germany, October 23-26, 2023*, Jason Flinn, Margo I. Seltzer, Peter Druschel, Antoine Kaufmann, and Jonathan Mace (Eds.). ACM, 182–199. https://doi.org/10.1145/3600006.3613170

[14] Qiumin Xu, Huzefa Siyamwala, Mrinmoy Ghosh, Tameesh Suri, Manu Awasthi, Zvika Guz, Anahita Shayesteh, and Vijay Balakrishnan. 2015. Performance analysis of NVMe SSDs and their implication on real world databases. In *SYSTOR*. ACM, 6:1–6:11.

[15] Zhuolong Yu, Yiwen Zhang, Vladimir Braverman, Mosharaf Chowdhury, and Xin Jin. 2020. NetLock: Fast, Centralized Lock Management Using Programmable Switches. In *Proceedings of the Annual Conference of the ACM Special Interest Group on Data Communication on the Applications, Technologies, Architectures, and Protocols for Computer Communication* (Virtual Event, USA) *(SIGCOMM '20)*. Association for Computing Machinery, New York, NY, USA, 126–138. https://doi.org/10.1145/3387514.3405857

[16] Xinjing Zhou, Joy Arulraj, Andrew Pavlo, and David E. Cohen. 2021. Spitfire: A Three-Tier Buffer Manager for Volatile and Non-Volatile Memory. In *SIGMOD '21: International Conference on Management of Data, Virtual Event, China, June 20-25, 2021*, Guoliang Li, Zhanhuai Li, Stratos Idreos, and Divesh Srivastava (Eds.). ACM, 2195–2207. https://doi.org/10.1145/3448016.3452819

[17] Tobias Ziegler, Carsten Binnig, and Viktor Leis. 2022. ScaleStore: A Fast and Cost-Efficient Storage Engine using DRAM, NVMe, and RDMA. In *SIGMOD '22: International Conference on Management of Data, Philadelphia, PA, USA, June 12 - 17, 2022*, Zachary G. Ives, Angela Bonifati, and Amr El Abbadi (Eds.). ACM, Philadelphia, PA, USA, 685–699. https://doi.org/10.1145/3514221.3526187

[18] Tobias Ziegler, Viktor Leis, and Carsten Binnig. 2020. RDMA Communciation Patterns. *Datenbank-Spektrum* 20, 3 (2020), 199–210. https://doi.org/10.1007/S13222-020-00355-7

[19] Tobias Ziegler, Jacob Nelson-Slivon, Viktor Leis, and Carsten Binnig. 2023. Design Guidelines for Correct, Efficient, and Scalable Synchronization using One-Sided RDMA. *Proc. ACM Manag. Data* 1, 2 (2023), 131:1–131:26.