NVMe-oAF: Towards Adaptive NVMe-oF for IO-Intensive Workloads on HPC Cloud

Arjun Kashyap
akashyap5@ucmerced.edu
University of California, Merced
Merced, United States

Xiaoyi Lu
xiaoyi.lu@ucmerced.edu
University of California, Merced
Merced, United States

ABSTRACT

Applications running inside containers or virtual machines, traditionally use TCP/IP for communication in HPC clouds and data centers. The TCP/IP path usually becomes a major performance bottleneck for applications performing NVMe-over-Fabrics (NVMe-oF) based I/O operations in disaggregated storage settings. We propose an adaptive communication channel, called NVMe-over-Adaptive-Fabric (NVMe-oAF), that applications could leverage to eliminate the high-latency and low-bandwidth incurred by remote I/O requests over TCP/IP. NVMe-oAF accelerates I/O intensive applications using locality awareness along with optimized shared memory and TCP/IP paths. The adaptiveness of the fabric stems from the ability to adaptively select shared memory or TCP channel and further applying optimizations for the chosen channel. To evaluate NVMe-oAF, we co-design Intel's SPDK library with our designs and show up to 7.1x bandwidth improvement and up to 4.2x latency reduction for various workloads over commodity TCP/IP-based Ethernet networks (e.g., 10Gbps, 25Gbps, and 100Gbps). We achieve similar (or sometimes better) performance when compared to NVMe-over-RDMA by avoiding the cumbersome management of RDMA in HPC cloud environments. Finally, we also co-design NVMe-oAF with H5bench to showcase the benefit it brings to HDF5 applications. Our evaluation indicates up to a 7x bandwidth improvement when compared with the network file system (NFS).

CCS CONCEPTS

• **Information systems → Network attached storage**; *Cloud based storage*; *Flash memory.*

KEYWORDS

NVMe-over-Fabrics, Shared memory, SPDK, HPC Cloud

ACM Reference Format:
Arjun Kashyap and Xiaoyi Lu. 2022. NVMe-oAF: Towards Adaptive NVMe-oF for IO-Intensive Workloads on HPC Cloud. In *Proceedings of the 31st Int'l Symposium on High-Performance Parallel and Distributed Computing (HPDC '22), June 27-July 1, 2022, Minneapolis, MN, USA.* ACM, New York, NY, USA, 15 pages. https://doi.org/10.1145/3502181.3531476

1 INTRODUCTION

Motivation: Solid-State Drive (SSD) are becoming quite popular in high performance computing (HPC) and cloud computing environments due to their low access latency when compared to disks [50, 51, 79]. To maintain high utilization and lower costs for SSDs, storage disaggregation is widely used [28, 45, 85]. NVMe-SSDs already suffer from long tail [21, 30] and unpredictable I/O latency [43], while separating compute and storage in disaggregated settings makes matter worse due to the introduction of network overheads [29]. In the past, Internet Small Computer Systems Interface (iSCSI) [37, 40] was the de facto mechanism for accessing remote hard disks but recently a new protocol, NVMe-over-Fabrics [9] (NVMe-oF), for flash disaggregation, is starting to gain traction in HPC and cloud computing environments [17, 28, 61].

HPC is widespread for solving scientific research problems in various domains like simulations, Big Data, and Artifical Intelligence (AI). With the emergence of cloud computing, researchers rushed to analyze the benefits of running HPC applications on cloud architecture, giving rise to a new and exciting research space, HPC cloud [16, 23, 25–27, 54]. As storage disaggregation and virtualization is common in data centers, most I/O requests issued by an HPC application running in a cloud get transformed into remote I/Os. Hence, network and remote I/O performance characteristics play a crucial role in the optimization of HPC applications running in the cloud [27]. This motivates us to find a solution that alleviate network transport overhead for remote storage access and achieves optimal I/O performance. Our goal is to improve data movement for I/O-intensive workloads in HPC clouds equipped with NVMe-SSDs.

To this end, we transform NVMe-oF into an adaptive fabric (AF), which eliminates network bottleneck for intra-node data movement using shared memory and optimizes inter-node data movement. We argue that this is crucial for four main reasons: a) prior literature [53] indicates that good portion of application I/O are completed by co-located storage services, b) intra-node storage and communication optimization is critical for achieving exascale computing goals [31, 80], c) existing NVMe/TCP has a scope for further improvement, and d) a lower amount of network communication abides in better performance for HPC applications in the cloud [20, 52, 55]. Therefore, we propose NVMe-over-Adaptive-Fabrics (NVMe-oAF) which HPC applications can utilize to accelerate remote I/Os over our adaptive fabric. Figure 1 illustrates the interaction of applications with storage services in the typical HPC Cloud architecture. Applications run in containers or virtual machines (VMs) and could be connected to the storage services that expose the remote SSDs over different network transports like TCP/IP over Ethernet and/or Remote Direct Memory Access (RDMA) with InfiniBand or RDMA over Converged Ethernet (RoCE).

Limitation of state-of-art approaches: The iSCSI protocol has known to be a bottleneck for remote I/O [44, 45]. Distributed file systems like GFS and HDFS are fine-tuned for large I/O sizes on storage devices but not for smaller sized read/write I/Os [24, 63].

HPDC '22, June 27-July 1, 2022, Minneapolis, MN, USA
© 2022 Copyright held by the owner/author(s).
ACM ISBN 978-1-4503-9199-3/22/06.
https://doi.org/10.1145/3502181.3531476

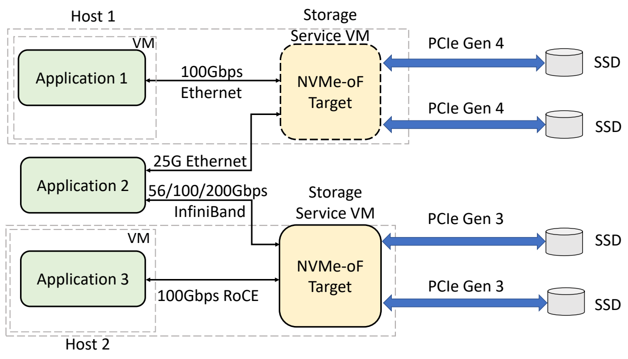


Figure 1: An example of HPC Cloud architecture considered in this study. Applications and storage service could either run in VMs/containers.

Other remote storage stacks like ReFlex [45] and i10 [33] that allow block-level remote access to SSDs are not deployed universally unlike the standardized NVMe-oF. Though NVMe-oF allows applications to take advantage of the NVMe [8] protocol, there is still a huge scope to improve the performance of accessing remote SSDs. The major drawbacks of NVMe-oF protocols that limit their performance and widespread adoption in HPC cloud are as follows. First, the networking fabric is slow [26, 54]. For example vanilla TCP/IP, and as stated before could adversely impact performance of HPC cloud applications. Second, using faster networks like InfiniBand could alleviate the slow network problem but they are difficult to manage in virtualized environments. Existing RDMA virtualization solutions [11, 49, 56, 67] are not mature enough [41] yet due to difficulty in orchestration, lack of migration support, high memory registration overheads, and so on [32].

Key insights and contributions: This paper proposes a novel concept, NVMe-oAF, which is inspired by HPC runtimes like MPI to accelerate the I/O data path for an application using shared memory while using the already existing TCP connection in the control path. Currently, NVMe-oF protocol lacks support for a shared memory channel and enabling it creates room for further optimization. A key insight for achieving high performance is NVMe-oAF accelerates I/O intensive applications using locality awareness along with optimized shared memory and TCP/IP paths. The adaptiveness of the fabric stems from the ability to adaptively select shared memory or TCP channel and further applying optimizations for the chosen channel.

For container-based applications, our proposed NVMe-oAF design can support Inter-Container Shared Memory [76] (ICSHMEM) channel, which is similar to host-level shared memory. For VM-based applications, our proposed design can leverage inter-VM shared memory channel, like IVSHMEM [6]. These two channels can be enabled and adaptively selected in our NVMe-oAF design once they are properly set up for containers and VMs. Note that our proposed designs can significantly improve the performance of NVMe-oF protocols while conforming to NVMe standards. To the best of our knowledge, this is the first work to enable shared memory and adaptive fabric schemes for NVMe-oF in the HPC cloud architecture.

We summarize the key contributions of our work as follows:

- Characterization of existing NVMe-oF network transports like TCP/IP and RDMA in HPC cloud environments.
- Enabling shared memory fabric in NVMe-oF for intra-node data movement and applying important optimizations like lock-free double buffer scheme, shared memory based flow control, and zero-copy transport.
- Improving the performance of NVMe/TCP in Intel Storage Performance Development Kit [72] (SPDK) for inter-node data movement with adaptive busy poll sockets and application-level chunk size detection.
- Using locality awareness to guide I/Os adaptively in selecting optimized shared memory or TCP/IP channels.
- Co-designing SPDK and Hierarchical Data Format Version 5 (HDF5) [1] storage runtimes to use our proposed adaptive fabric for communication, buffer allocation and management, which allows applications to take advantage of NVMe-oAF.
- Extensively evaluating NVMe-oAF with SPDK microbenchmarks and HDF5 application-level benchmarks show that our design can significantly improve the performance of NVMe-oF protocols in terms of latency, tail latency, throughput, and concurrency.

Experimental methodology: To evaluate our proposed NVMe-oAF designs, we run Intel SPDK based various workloads over virtual machines. The NVMe-oF target runs in one VM and acts as the storage service exposing NVMe-SSD over the network for applications. We use the SPDK's standard benchmark tool, perf [12], for running microbenchmarks instead of FIO [15] as it has higher overhead when compared to SPDK perf [34]. We also evaluate NVMe-oAF with h5bench [47] I/O kernels to gauge the performance improvement it brings to real-world HPC workloads in cloud environments.

Regarding fabrics, we choose to evaluate with different generations of high-speed Ethernets (10/25/100 Gbps) and RDMA networks (56Gbps InfiniBand and 100Gbps RoCE). Note that according to the latest Top500 [3] list, 10/25Gbps Ethernet networks are still the most popular interconnect technology (28.4%) being used in high-end machines and we believe that many machines will continue to use 10/25Gbps Ethernet for years. Thus, evaluation of NVMe/TCP over 10/25Gbps is still extremely crucial.

Note that our extensive evaluations show that our proposed NVMe-oAF design can achieve up to **7.1x** bandwidth and up to **4.2x** latency improvements for various workloads over commodity TCP/IP based 10Gbps and 25Gbps networks. We are also able to achieve comparable (and sometimes better) performance to NVMe-over-RDMA by avoiding the cumbersome management of RDMA in HPC cloud environments. Lastly, to view the benefits of our design over real applications in supercomputing systems, we co-design NVMe-oAF with h5bench as it contains representative HDF5 benchmarks for popular applications. Our results demonstrate that h5bench's write/read kernel achieves up to **7x** bandwidth improvement while using NVMe-oAF in comparison to NFS [62].

Limitations of the proposed approach: We assume a standard security model where the physical node cannot be comprised. Each application and NVMe-oF target have a separate shared memory channel to avoid any malicious application snooping on shared memory fabric. Further discussion on security is provided in Section 6.

2 BACKGROUND

In this section, we present some necessary background information for this paper.

2.1 NVMe-over-Fabrics

NVMe-over-Fabrics [9] (NVMe-oF) is a protocol specification that allows clients to speak to remote NVMe-SSDs over some kinds of fabrics. NVMe-oF lets client use Non-Volatile Memory Express [8] (NVMe) protocol, like in the case of local SSD, to transfer data between disaggregated storage. NVMe-oF has a one-to-one mapping between I/O submission queues and I/O completion queues and builds on the NVM subsystem architecture that presents a collection of controllers which are used to access namespaces [2]. Current transports supported by it are Fibre Channel, Ethernet, InfiniBand, and RDMA. To the best of our knowledge, a shared memory based channel is missing in NVMe-oF which we introduce and discuss in detail in Section 4. NVMe-oF specification names the node that is attached to SSD via PCIe and exposes the SSD as block device over the network as NVMe-oF target while the node initiating I/O as NVMe-oF client/initiator.

2.2 Intel SPDK

Intel's SPDK library consists of tools and drivers that allow applications to achieve high I/O performance. It operates in userspace to avoid expensive syscalls and in lockless mode by pinning its connections to cores. SPDK lowers latency by polling for I/O completions rather than event-based at the expense of high CPU utilization. The high performance of SPDK can be attributed to userspace and asynchronous NVMe driver written as a C library that provides highly parallel access directly to an SSD from a userspace application [13]. It also consists of a userspace application, NVMe-oF target, which exposes block devices over the network. There are two major drawbacks in Intel SPDK's NVMe/TCP implementation. One is that the NVMe/TCP uses interrupts which conflicts with the polling-based design of SPDK. Second is that the NVMe/TCP stack in SPDK is not optimized for intra-node communication. Thus, the stock NVMe/TCP in SPDK has sub-optimal performance which we characterize further in the following Section 3.

2.3 SR-IOV, IVSHMEM, and ICSHMEM

Single Root I/O Virtualization [11] (SR-IOV) interface allows separation of resources for PCIe adapters and gives the impression of availability of multiple PCIe devices. It consists of physical and virtual functions (VFs) where VFs are allowed to be pass-throughed to the VMs. Thus, the benefit of SR-IOV is its ability to efficiently virtualize interconnect resources among VMs. Inter-VM Shared Memory [6] (IVSHMEM) allows the host's shared memory region to be available in VMs as a virtualized PCI device. Hence, co-located VMs can communicate with each other over shared memory. Inter-Container Shared Memory (ICSHMEM) can be achieved by enabling sharing of Inter-Process Communication (IPC) namespace among containers or with the host [76]. The benefits of IVSHMEM and ICSHMEM is that they can provide better communication performance when compared to TCP/IP stack.

3 PERFORMANCE CHARACTERIZATION

In this section, we present our performance analysis of existing NVMe-oF schemes, which will further demonstrate the motivation of this paper.

3.1 Performance of Existing NVMe-oF Schemes

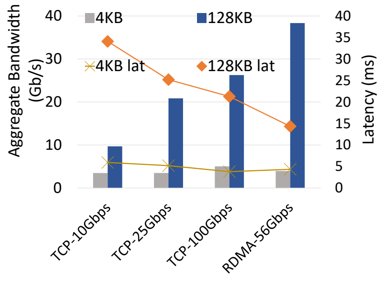
To compare the bandwidth and latency characteristics of existing NVMe-oF protocols over Ethernet (10 Gbps, 25 Gbps, and 100Gbps) and RDMA-over-InfiniBand (IB FDR, 56 Gbps) with SR-IOV support [11], we mimic the common architecture where multiple applications use the persistent storage exposed via storage services. The storage service runs on the NVMe-oF target device whose backend is commonly a collection of SSDs. Thus, in our characterization a single NVMe-oF target device is attached to up to four NVMe-SSDs and each SSD is exposed via a storage service, which leads to increased storage bandwidth when compared to the network.

In this experiment, we ask four applications to issue sequential read and write commands to four NVMe-SSDs over the same TCP/IP and RDMA channels on a single physical host. Each application sends I/O requests to one NVMe-SSD following a one-to-one mapping between the client and target SSD, which can avoid the possible resource contention and provide better performance. More details about our experiment setup can be found in Section 5.1. Then, we measure the aggregate bandwidth and average latency for TCP-over-10Gbps, TCP-over-25Gbps, TCP-over-100Gbps, and RDMA-IB-FDR transports for 4KB and 128KB I/O size as shown in Figure 2. We omit the results for other I/O sizes due to brevity.

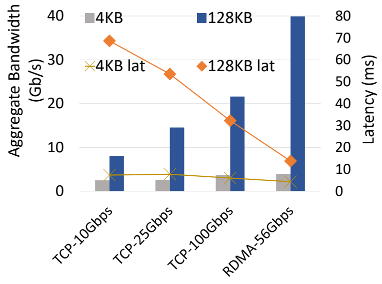
For 10Gbps Ethernet, we can see that network bottleneck does not allow the clients to utilize the available storage bandwidth for any workload. Also, the network bandwidth is not fully utilized for any of the workloads over 25Gbps/100Gbps Ethernet. On the other hand, NVMe/RDMA has larger network bandwidth for both read and write workloads. From this characterization we observe major performance bottleneck in the existing NVMe-oF protocols in the presence of multiple applications. There is a steep performance gap between NVMe/TCP and NVMe/RDMA. The bandwidth gaps between NVMe/TCP-100Gbps and NVMe/RDMA for four NVMe-SSDs are 1.85x and 1.46x in terms of peak write and read bandwidth, respectively, while the average latency follows the general trend of RDMA being faster than TCP with latency increasing with higher I/O size. These interesting performance characteristics of existing NVMe-oF protocols motivate us to perform further performance analysis in the following section.

3.2 Analysis of Existing NVMe-oF Schemes

Here we try to discover which components consume the majority of the time for the end-to-end journey of a remote I/O request as observed by the client/application. For this analysis, we break down the average latency observed by the application into three parts. The first part is the time remote SSD takes to execute an I/O request submitted by NVMe-oF target, called "I/O time". Second is the "communication time (comm. time)" that indicates the time the I/O request spent in transit or in the network. We call remainder of



(a) Sequential Read



(b) Sequential Write

Figure 2: Performance evaluation for existing NVMe-oF transports

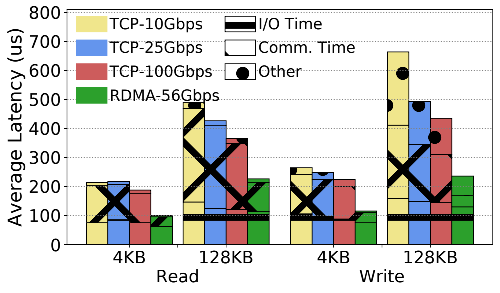


Figure 3: Sequential Read/Write latency breakdown over different NVMe-oF protocols

the time as "other" which might go into preparing and processing of the NVMe-oF I/O request by client and target.

The latency breakdown for 4KB and 128KB I/O sizes is shown in Figure 3 and the setup is similar to the one in Section 3.1. Clearly, high communication time is the major culprit for the stark difference between the aggregate bandwidth of NVMe/TCP when compared to NVMe/RDMA. Another important observation from these figures is that write operations for higher I/O sizes in NVMe/TCP spend significant time in the "other" portion of the request when compared to read operations. This is because for write operations, the application needs to first fill and then copy out the buffer containing payload (considered as processing/preparation time at the client) which is then transported to the target. Whereas for read operations, the client only needs to have the buffer space available to hold/read the payload sent by target. This behavior is not seen in NVMe/RDMA as the target directly has access to the client's buffer. We use this key insight for our NVMe-oAF design during buffer creation/allocation in shared memory (explained further in Section 4.4) to reduce the request processing/preparation overheads for write I/O. NVMe-oAF further reduces the number of buffer copies when compared to NVMe/TCP with buffers as discussed in Section 4.4.3.

At small I/O sizes (4KB), the "I/O time" is the major bottleneck for NVMe/RDMA read commands but this soon changes. When four clients perform 128KB reads from four SSDs over NVMe/RDMA the ratio of communication time to I/O time is 1 : 1.11. This explains the aggregate read bandwidth degradation for multiple streams/SSDs as seen in Figure 2(a). Later, we will see that unlike NVMe/RDMA, NVMe-oAF is able to achieve higher read aggregate bandwidth as network is no longer a bottleneck (Figure 11(a)).

3.3 Summary of Characterization

Thus, there are two major problems in the existing NVMe-oF schemes for I/O-intensive applications. One is the huge performance gap between NVMe/TCP and NVMe/RDMA schemes. This issue is further exacerbated by the diverse workloads supported by applications that consist of varying I/O sizes and concurrencies. The second problem is the complexity [48] and inconvenience to manage [32] RDMA in HPC cloud environments which hinders users from utilizing NVMe/RMDA. Even though we are not focusing on the RDMA design path, we still report its performance.

Prior research [20, 52, 55] shows that lowering network communication helps improve performance of HPC applications running in the cloud. Hence, these studies along with our findings in the above discussions lead to an interesting and useful question: *Is there a fabric that can provide high bandwidth and low latency like NVMe/RDMA, reduce network communication, and at the same time be easy to manage in HPC cloud environments?* The leads us to the creation of a new NVMe-oF fabric and runtime for co-located applications that should be application oblivious, easy to manage, and provide high-performance. In this context, this paper proposes NVMe-oAF that further optimizes NVMe/TCP using a shared-memory channel, adaptive busy poll sockets, and chunk size selection in a cost-effective way. Our evaluations show that optimization of the shared memory channel is non-trivial and allows applications to achieve significant performance improvement over NVMe/TCP and offers comparable performance to NVMe/RDMA.

4 DESIGN

This section presents our designs of NVMe-oAF, a combination of TCP/IP and shared memory, which can be used to accelerate I/O operations for HPC applications. These applications generally use TCP/IP as the mode of communication. Thus, applications running I/O intensive workloads experience high latency and low bandwidth due to NVMe/TCP. Figure 4 showcases our design on turning NVMe-oF into an adaptive fabric. Adaptive Fabric (AF) consists of three major components and two kinds of optimized channels - shared memory and TCP/IP. First is the *Connection Manager*, which is responsible for establishing an adaptive fabric channel between NVMe-oF client and target and reclaiming resources at the end. The second component, *Buffer Manager*, allocates buffer either in shared memory or Data Plane Development Kit (DPDK) [4] memory pool based on the fabric and re-uses it when possible. The third component is the *Locality Awareness* which determines whether NVMe-oF client and target are located on the same node and performs shared memory mapping.

NVMe-oAF enhances NVMe/TCP performance for inter-node communication by tuning the application-level chunk size for I/O requests and using busy polling to reduce network latency. To further speed up the I/O performance for intra-node communication we need another fabric, shared memory in this case. The key idea of AF is that the data path uses fast fabric/transport, shared memory, and the control operations travel through TCP/IP. This significantly

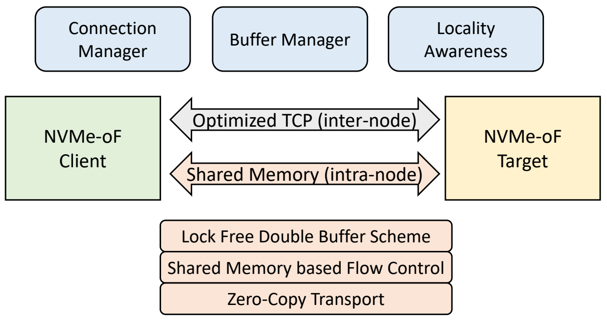

Connection Manager
Buffer Manager
Locality Awareness
NVMe-oF Client
Optimized TCP (inter-node)
Shared Memory (intra-node)
NVMe-oF Target
Lock Free Double Buffer Scheme
Shared Memory based Flow Control
Zero-Copy Transport

Figure 4: Overview of NVMe-oAF architecture. Additional components are added on top of NVMe-oF by adaptive fabric.

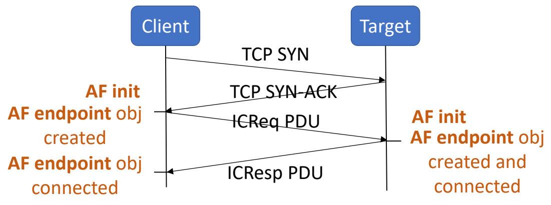


Figure 5: Connection Establishment between NVMe-oAF client and target.

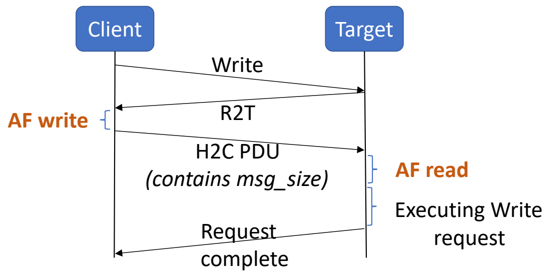


Figure 6: Out of band message notifications for a single write command over TCP/IP. Read is similar to write.

increases the bandwidth and lowers the latency of I/O operations as the I/O payload is transferred over the shared memory rather than TCP/IP. A naive solution would be to enable a shared memory channel between the application and storage service for the data path. This does increase the bandwidth when compared to NVMe/TCP but is not the optimal solution because the flow control needed for TCP/IP does not apply to the shared memory channel. We show in later sections how a shared memory aware flow control, zero-copy transport, and a lock-free double buffer design achieve higher bandwidth and lower latency compared to a naive NVMe-over-Shared Memory approach. Other advantages of using AF for I/O operations are uniform interface for NVMe applications and exploiting HPC technologies. The remainder of this section discusses each of the above components in detail.

4.1 Connection Establishment and Buffer Management

The connection establishment between the target and client happens out-of-band over NVMe/TCP connection using Protocol Data Units (PDUs) as described in NVMe-oF specification [9] with the aid of Connection Manager (CM). As seen in Figure 5, the client first establishes a TCP connection using TCP 3-way handshake and initializes AF. The CM creates the AF endpoint object on the client and exchanges connection configuration parameters through Initialize Connection Request (ICReq). On the target side, CM also initializes adaptive fabric, creates and connects its AF endpoint object. Then the target responds back with Initialize Connection Response (ICResp) PDUs which allows the client to connect its own AF endpoint object as well. Once the AF endpoint object of both client and target are connected, data can be exchanged between them.

Buffer Manager allocates buffer either in shared memory or DPDK memory pool based on the locality discussed in Section 4.2 during connection establishment. Its responsibilities during the entire I/O process include buffer creation, alignment, formatting, reuse and reclamation. The Buffer Manager further implements a lock-free double buffer scheme and provides APIs for zero-copy transports for the shared memory region discussed in Sections 4.4.1 and 4.4.3, respectively.

Figure 6 highlights the out-of-band message notifications and data transfers over AF for a single write I/O command issued by the client. The message notifications (indicated via arrows) occur out-of-band over TCP/IP while the actual data is transported over shared memory.

4.2 Locality Awareness

As it is possible that multiple client applications could talk to the same storage service, we assign each client a different shared memory region keeping security in mind. Since clients and targets could be located on different nodes, locality detection is vital to determine the availability of a shared memory channel between them. Whenever a new client application wishes to interact with the storage service on the same host, we hotplug the shared memory region to both the client and storage service and will later exchange the relevant information about it over existing TCP channel. This hotplug mechanism is implemented in an out-of-band fashion, which means we need some proxy or helper process running on the host to enable it. In practice, this helper process can be the HPC/cloud resource manager, such as Kubernetes or OpenStack or SLURM.

Once the helper process attaches an IVSHMEM or ICSHMEM region to the virtual machine or container, NVMe-oAF enabled client and target will be notified by the helper process through a pre-reserved shared memory region. The Connection Manager monitors the flag in a pre-reserved shared memory region periodically in both NVMe-oAF client and target processes. The detected flag will be used for locality checking during the connection establishment process, which has been discussed in Section 4.1. The locality detection process can be easily simplified in real HPC/cloud deployments since the locality information can be attained from the resource managers.

Now selecting the appropriate channel adaptively for each I/O request is imperative to allow applications to fully exploit the high bandwidth and low latency characteristics of adaptive fabric. Based on the locality of both client and target, whether present on the same physical node or not, they agree to use shared memory channel along with TCP/IP. The initialization requests between client and target always go through TCP/IP. Before writing to or reading from the AF, AF endpoint object is checked to learn about the completion of the AF initialization phase and shared memory mapping. In the presence of a shared memory channel, for the workload requests, the adaptive fabric's channel selection kicks in by cleverly decoupling the payload from the requests. By removing the payload from the I/O request, the request consists of only the control message which is transmitted out-of-band over TCP/IP. Meanwhile, the large sized I/O payloads are transported over the shared memory.

4.3 NVMe-oSHM: NVMe-over-Shared-Memory

NVMe-over-Shared-Memory uses our newly proposed shared memory fabric for the data path and TCP/IP for control path between the client and the target. In case the client and target do not reside on the same node (no shared memory channel), our adaptive fabric scheme automatically picks up NVMe/TCP as described in Section 4.2.

Figure 7 shows the journey of a write I/O command over the adaptive fabric scheme utilizing both TCP and shared memory channel. Here, connection establishment between the client and the target occurs out-of-band over TCP/IP (as explained in Section 4.1) and not shown in the Figure 7. From the Figure 7, in step ①, client application issues a 16KB write I/O request to the storage service or target. The target allocates a DPDK buffer to receive the payload and responds back with a Ready to Transfer (R2T) PDU in step ②. Upon receiving the R2T PDU, the client copies the payload of the write request (16384 bytes in this case), along the the I/O vector (16 bytes) pointing to this payload to the shared memory region via AF write as step ③. After the payload is written to shared memory, client sends the location and size of payload as a Host-to-Controller (H2C) notification to the target in step ④. In step ⑤, target copies the data into the DPDK buffer based on the metadata received from the out-of-band H2C PDU. Now, the request is ready to be submitted to the NVMe-SSD in step ⑥. When the write request is executed on the NVMe-SSD, step ⑦, a request completion notification is delivered to the client in step ⑧. The buffers can be re-used/reclaimed before the next I/O command.

4.4 Optimized Designs for NVMe-oSHM

4.4.1 Lock-free double buffer scheme. As the application would perform both read and write operations, the shared memory channel should be able to support bi-directional communication or data transfer. We achieve this by treating the shared memory region as a double buffer and ensuring all reads from/writes to shared memory occurs in a lock-free way. This eliminates the need to acquire a lock on shared memory each time we need to read/write from/to it. First, we logically partition the entire shared memory region into two buffers, one for client and another for the target. Second, to permit high concurrency for I/O operations we ensure that the payload/data is written to/read from an appropriate offset in shared

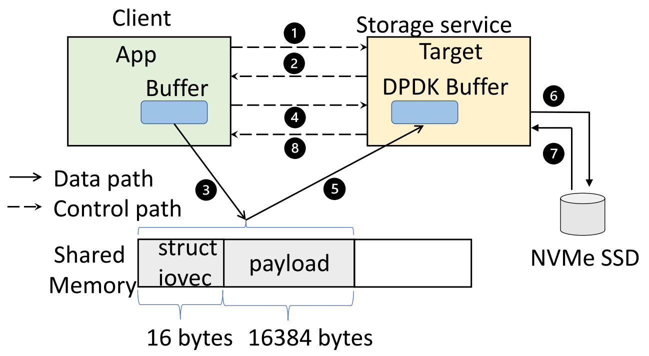


Figure 7: Overview of NVMe-over-Adaptive-Fabirc (NVMe-oSHM channel) for a single write request. The control path is over TCP/IP and the data path is over shared memory.

memory, i.e., both the client and target buffer are logically divided into slots. Each logical slot is equal to the I/O size and the total number of slots are the same as the queue depth. The offset/slot in client and target buffer is chosen in a round-robin fashion with respect to the application I/O depth. A slot/offset is computed before each I/O request copies data to shared memory. When the payload/data is copied to a particular offset in shared memory region, this offset is sent over to client/target as an out-of-band notification based on the I/O type. Thus, our scheme ensures both the client and the target write to/read from separate portions of shared memory channel during pure or mixed workloads in a lock-free manner.

4.4.2 Shared memory based flow control. Presently, NVMe/TCP has two types of flow control for write operations based on the I/O sizes. Small I/O sizes (<8KB) follow the in-capsule data flow wherein the payload is transferred along with the write I/O command. This flow assumes that the NVMe-oF target would have sufficient buffer capacity to capture the I/O request along with the payload. With I/O sizes >8KB, a more conservative flow control method is employed, where the write request and the payload are exchanged within multiple requests between the client and target, which is similar to the flow as shown in Figure 7. Hence, the main difference between the two flow control approaches is the number of control messages. In in-capsule data flow, just one message is enough for the target to receive the request and submit the I/O to the NVMe-SSD. But in the conservative flow control method, three messages are exchanged before the write I/O could be submitted to the SSD.

When using shared memory for transferring the payload, the data could reside in the shared memory until the the target is ready to read and process the payload unlike TCP/IP channel. Hence, the flow control mechanism can adaptively be changed from a conservative one to the in-capsule based when shared memory channel is available. This optimization reduces the out-of-band control messages exchanged for every single write I/O command irrespective of the I/O size. For example, in Figure 7, our shared memory based flow control eliminates two control messages among the four control messages exchanged for each I/O operation. Particularly, our shared memory based flow control would eliminate steps ② and ④

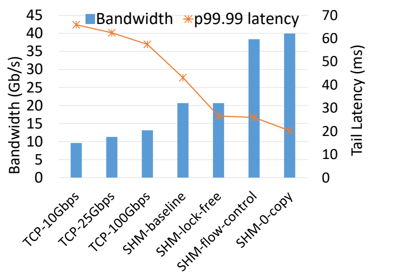


Figure 8: Bandwidth evaluation of various NVMe-oAF design optimizations for sequential read workload with I/O size 512KB.

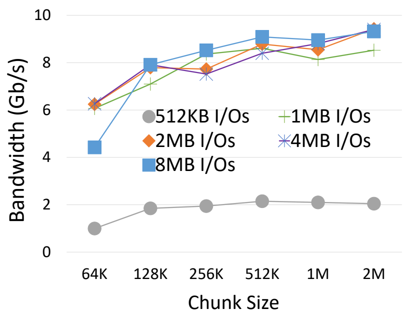


Figure 9: Finding optimal chunk size for NVMe/TCP (over 25Gbps) for random read I/Os. We observed similar results for TCP-100Gbps.

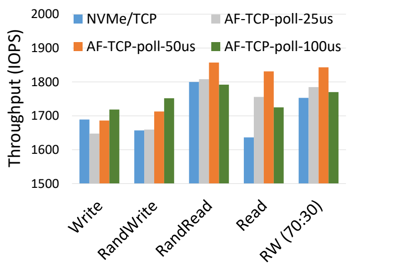


Figure 10: Throughput comparison of NVMe/TCP (over 10Gbps) and AF (in TCP only mode) for various busy polling rates for 128KB I/O.

and interchange steps ① and ③ keeping the rest of the I/O flow the same.

4.4.3 Zero-copy transport. Naively, using shared memory mechanism to transfer data/payload does not allow the application to reach peak performance. Consider a naive shared memory approach for write I/O over shared memory. When the client is ready to transfer the payload, it copies the data from its buffer to the shared memory and sends an out-of-band notification to the target. As the target receives this notification, it copies the payload from the shared memory to its own buffer and starts to process the I/O command. Thus, the entire process involves one copy of payload/data from the client's buffer to shared memory. This naive design limits the NVMe-oSHM to achieve maximum throughput and low latency. An important thing to remember here is that the copy from shared memory to the target's buffer cannot be avoided as this buffer in SPDK is managed by DPDK to allow direct memory access for NVMe devices from userspace [5].

Hence, to realize the full potential of shared memory in terms of performance, we implement a zero-copy approach by eliminating the one copy from client's buffer to shared memory transport. By co-designing the application or upper-layer runtime with NVMe-oAF, our Buffer Manager component creates application buffers directly on shared memory. When the client's buffer reside on shared memory, we can save the extra copy overhead as the target could directly read from/write to client's buffer developing a zero-copy shared memory transport.

4.4.4 Analysis of design optimizations. Here we would like to quantify the benefits of the successive design optimizations on the performance of NVMe-oSHM as shown in Figure 8. The "SHM-baseline" represents a naive shared memory design to transfer payload between client and target rather than using TCP sockets. It uses locks as a way to access the shared memory region. Despite having locks for accessing a shared resource, its bandwidth is 1.83x better than NVMe/TCP-25Gbps. The "SHM-lock-free" design removes the use of locks to access shared memory by using the lock-free double-buffer design discussed in Section 4.4.1. Though this design does not improve the bandwidth, it drastically reduces the tail latency (p99.99) by 38% which is extremely important for latency-critical applications. Next, by removing extra control messages and optimizing the flow control we are able to increase the bandwidth by 1.85x. Finally, in our "SHM-0-copy" scheme we eliminate the extra copy from client's buffer and lower tail latency by 22%. As "SHM-0-copy" contains all the optimizations and shows the best performance, we choose this design in our experimental evaluation.

4.5 TCP-channel Optimization

We propose two optimizations to NVMe/TCP in our AF design to further improve the performance of TCP channel. First, NVMe/TCP statically sets the application-level chunk size to 128KB. Based on the chunk size, I/O requests are internally broken down to $\left\lceil \frac{I/Osize}{chunk-size} \right\rceil$ requests and increase the number of I/O requests for large-sized I/O. Chunk size also is used to create memory buffers/pools on NVMe-oF target. As seen in Figure 9, we vary the chunk size and measure the bandwidth of various sized I/O streams. If the chunk size is large (say 2M), small-sized I/Os lead to under-utilization of memory but choosing a very low chunk size hurts bandwidth. We find that the 512KB chunk size is ideal for 25Gbps Ethernet as it provides close to the highest bandwidth and at the same time keep the memory utilization to a minimum for all the I/O streams. Hence, optimal chunk size can be adaptively chosen based on underlying hardware architecture.

Second, we use busy polling to reduce the network latency by continuously polling the network queues. The amount of time kernel spends in polling is an important factor in determining the application performance as seen in Figure 10. We observe that statically allocating busy polling time is not an optimal solution due to workload variability. For example, sequential write workloads when polled for shorter duration (25us) lead to a decrease in the throughput, even in comparison to purely interrupt-based NVMe/TCP. This happens due to the latency of write operations being high and the short busy poll time adds extra overhead reducing the total throughput. Whereas, for a high busy poll time (100us), the pure write workloads have the highest throughput when compared to smaller busy poll times. On the other hand, read workloads achieve peak throughput when busy polling time is set between 25-50us. This is because read operations, in general, are faster than writes and high busy poll times degrades their performance. Thus, our design carefully selects the busy polling rate based on the type of workload to allow the client in attaining maximum throughput.

4.6 Adaptive Fabric based SPDK

In this section we discuss how SPDK utilizes adaptive fabric to benefit from different underlying fabrics like TCP and shared memory based on locality of the application and transport availability. First, we integrate our Connection Manager with SPDK using which the client establishes a TCP connection with the target. The CM creates the AF endpoint object that stores whether the adaptive fabric channel is initialized between the client and target as well as other information like availability of shared memory. Next, SPDK uses our Buffer Manager to allocate buffer either in shared memory or DRAM based on the AF endpoint context object with the requested size. Once the handshake and initialization are complete, SPDK writes to AF to send I/O to the client. The AF write distinguishes the control and data path during the runtime and sends the data over shared memory whereas the control messages over TCP, unbeknownst to the application. Similarly, the NVMe-oF target is modified to use the AF read to process both the out-of-band control notifications and payload transfers.

5 PERFORMANCE EVALUATION

5.1 Experimental Setup

We run our experiments on upto four nodes in Chameleon Cloud [39] and CloudLab [22]. Then on each node we create two virtual machines (VMs) using QEMU [10] emulator v2.0.0, one to run NVMe-oF client and other for SPDK NVMe-oF target. NVMe-oF target VM acts as a storage serivce while the NVMe-oF client VM could be considered to be any application talking to remote NVMe-SSD. Detailed information about the nodes and VMs is present in Table 1. The Chameleon Cloud node is equipped with Broadcomm 10Gbps NetXtreme II Ethernet interface and Mellanox 56Gbps FDR InfiniBand interface while the CloudLab node is equipped with Mellanox ConnectX-5 25 Gbps and Mellanox ConnectX-5 Ex 100 Gbps NIC. The VMs have access to InfiniBand and Ethernet interfaces via SR-IOV [11] so all communication traffic across VMs goes over real NIC through SR-IOV.

We also use QEMU [10] to emulate up to four NVMe-SSDs and attach them to NVMe-oF target VM and plain Inter-VM Shared Memory (IVSHMEM [6]) to create shared memory region between the client and target. We present NVMe-oF performance evaluations over TCP (TCP-10Gbps, TCP-25Gbps, and TCP-100Gbps) and adaptive fabric (TCP with shared memory, i.e., SHM-0-copy). "SHM-0-copy" incorporates all our design decisions mentioned in Section 4.4. We introduce results of NMVe/RDMA over InfiniBand and RoCE to indicate that our adaptive fabric design achieves comparable performance and at the same time avoids the tedious management of RDMA in HPC cloud environments. We could not compare AF performance with 100Gbps InfiniBand due to device unavailability on CloudLab and Chameleon Cloud currently. We ran into some issues to run SPDK with RoCE on the VMs. Thus, RoCE numbers were taken by running SPDK on two CloudLab physical nodes (configuration same as that in Table 1) directly connected by Mellanox ConnectX-5 Ex 100 Gbps NIC and accessing a real NVMe-SSD. Hence, NVMe/RoCE represents the upper bound of performance

	Physical Node	Client VM	Target VM
Processor	CC- Intel Xeon CPU E5-2670 v3 @ 2.30GHz CL- AMD EPYC 7402P @ 2.80GHz		
CPU(s)	48	14	14
NUMA (s)	2	1	1
DRAM	128GB	16GB	
Kernel	3.10.0-957.27.2.el7	3.10.0-1127.19.1.el7	
OS	CentOS Linux 7.7		
OFED	MLNX_OFED 5.0-1.0.0.0		
Scale	Upto 4 nodes		

Table 1: Experiment configuration (CC - Chameleon Cloud and CL - CloudLab)

an application could achieve as there is no virtualization layer overheads. As there was only one real NVMe-SSD on CloudLab machine, we were unable to collect results when four SSDs are communicating with four clients in case of NVMe/RoCE.

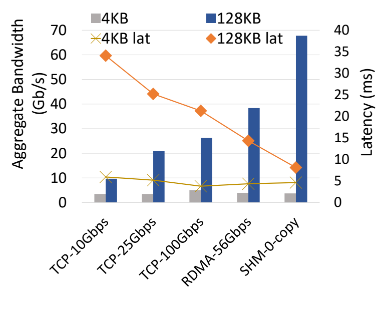
We simulate TCP-25Gbps by using IPoIB and throttle it down for TCP-10Gbps. Intel SPDK v20.07 is used to run NVMe-oF target and its perf tool is used as NVMe-oF client for the performance tests. SPDK has been modified to use the adaptive fabric for communication and buffer allocation as discussed in Section 4.6. The zero-copy design of NVMe-oAF is co-designed with applications, in this case with perf and h5bench, to display the full potential of adpative fabric for HPC applications.

For all the experiments, the queue or I/O depth is set to 128, the running time is 20 seconds, and one client/application communicates to one target device unless otherwise stated. With experiments involving multiple clients and remote SSDs, each NVMe-oF client and target are pinned to separate cores to avoid CPU being the bottleneck. For our evaluations with HDF5 benchmarks we use h5bench v1.0 and hdf5 v1.12.1. Each experiment is repeated five times and the average value is reported.

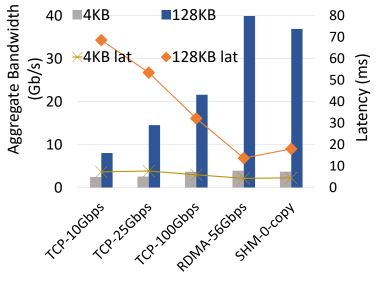
5.2 Overall Benefits

First, we present the performance benefits, Figure 11, of NVMe-oAF as compared to NVMe/TCP for four applications/streams communicating to four NVMe-SSDs (one-to-one mapping). In terms of peak read bandwidth, NVMe-oAF outperforms NVMe/TCP-10Gbps by 7.1x. The average latency of NVMe-oAF is also lower than that of NVMe/TCP. At 128KB I/O size, read latency of NVMe/TCP-10Gbps is 4.2x higher than that of NVMe-oAF while the write latency of NVMe/TCP-25Gbps is 2.97x higher than NVMe-oAF. When compared to NVMe/RDMA, NVMe-oAF outperforms NVMe/RDMA by 1.78x when performing 128KB reads from four SSDs.

Data center have variety of networking operating at different capacities. So, NVMe/TCP performance under various network speeds is equally important to understand their implications on applications. Besides, NVMe/TCP is the alternate mechanism of adaptive fabric when applications do not reside on the same host/node. From Figure 11(a) one can observe that the bandwidth attained by NVMe/TCP-25Gbps is similar to NVMe/TCP-10Gbps for 4KB workloads and only marginally better for 128KB sized workloads.



(a) Sequential Read



(b) Sequential Write

Figure 11: Performance evaluation of NVMe-oAF when four applications talk to four SSDs

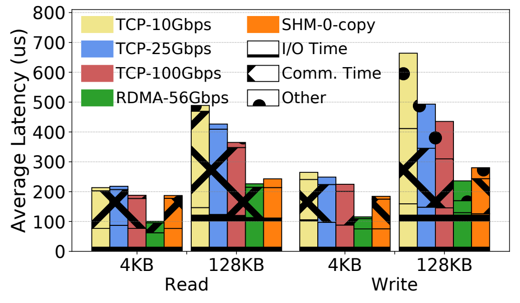


Figure 12: Sequential Read/Write latency breakdown of NVMe-oAF for four SSDs

At higher network speeds (TCP-100Gbps), the read and write bandwidth is 1.26x and 1.48x when compared to NVMe/TCP-25Gbps, respectively. Likewise, the TCP-10Gbps, TCP-25Gbps, and TCP-100Gbps have similar average latencies for 4KB I/O sizes.

5.3 Benefits Analysis

In Figure 12 we compare the latency breakdown for read and write commands with NVMe-oAF with four streams. A clear benefit of the zero-copy design and shared memory based flow control of NVMe-oAF is the reduction in communication time for 128KB I/O. The decrease in latency for I/O operations can be attributed to the elimination of an extra copy from the client's buffer to shared memory (zero-copy design) and cutting down on the number of control messages exchanged (shared memory based flow control) by NVMe-oAF. For read workloads with four applications/streams, NVMe-oAF is able to reduce average latency by 50%, 43%, and 33% when compared to NVMe/TCP-10Gbps, NVMe/TCP-25Gbps, and NVMe/TCP-100Gbps, respectively. Lastly, NVMe-oAF is also able to bring down the "other" component of average latency, which constitutes the client preparation time, for write I/O as the buffer resides on the shared memory saving buffer preparation/processing overheads as explained in Section 3.2.

5.4 Tail-latency Studies

In Figure 13 we investigate the tail latency characteristics for a mixed read-write (70:30) 128KB workload with different fabrics. The tail latency of NVMe/TCP-100Gbps is slightly lower than NVMe/TCP-25Gbps and NVMe/TCP-10Gbps but the tail latency of NVMe-oAF is 3x smaller than that of NVMe/TCP-100Gbps and NVMe/RDMA. Even though the average latency of NVMe/RDMA and NVMe/RoCE is lower than that of NVMe-oAF, its tail latency is quite high due to memory registration overheads incurred by RDMA. To understand this phenomenon we conduct the same experiment with a running time being 3-4 times higher than our initial experiment. As expected, we found the tail latency of NVMe/RDMA to be lower than that of NVMe-oAF. This confirms that memory registration overheads incurred by RDMA could impact the tail latency of short-running or latency-sensitive applications.

5.5 Concurrency

Next, we examine the effect of fabric upon the ability to exploit the inherent parallelism available in NVMe-SSDs to scale performance. In this experiment, we use a single NVMe I/O queue pair as the I/O submission path to NVMe-SSD with varying queue depth between 1 to 128. From Figure 14, we first observe that network speed does not help much in exploiting the concurrency of a single SSD. After queue depth of 8, bandwidth for NVMe/TCP and NVMe/RoCE remains almost constant indicating that higher queue depth has minimal impact on improving the bandwidth.

On the other hand, the lock-free double buffer design in "SHM-0-copy" allows NVMe-oAF to not hinder the linear scaling of bandwidth with increasing queue depths. At queue depth of 1, NVMe-oAF is not able to achieve significant performance due to the overhead in the control plane. This is also seen in Figure 12 where the communication time of AF is comparable to TCP due to control messages being predominant for small (4KB) I/O. At higher I/O size (128KB) and multiple streams, the communication time of NVMe-oAF and NVMe/RDMA are quite similar highlighting that control messages overhead decreases. Our AF scheme shows that control plane overheads are not negligible and opens a future research direction to reduce control plane latency by further optimizing it or by utilizing faster protocols like RDMA or user-level TCP/IP in the control path.

5.6 Different Workloads

Till now all our results only considered sequential workloads. It is equally important to understand the performance implications introduced by random workloads as different applications would produce a different mix of workloads over time. We choose three random workloads with varying proportions of read to write operations in order to simulate a read-heavy (95:5), equal (50:50), and write-heavy (5:95) nature of I/O. The throughput for all the fabrics with a single stream/SSD is shown in Figure 15. Again, the speed of TCP network has slight impact on the throughput for all the three workloads. In contrast, NVMe-oAF is able to achieve 2.33x improvement in throughput on average when compared to NVMe/TCP-100Gbps at 512KB I/O size. Also, NVMe-oAF has a modest decrease, 5-13.5%, in throughput when compared to NVMe/RDMA-56Gbps

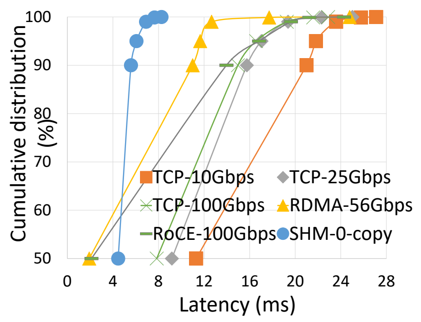


Figure 13: Tail latency for seq 128KB read-write (70:30) workload (Lower better).

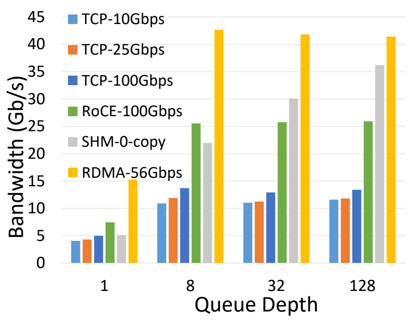


Figure 14: Concurrency estimation for sequential read workload at an I/O size of 128KB.

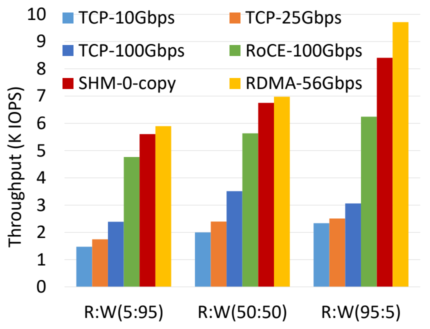


Figure 15: Throughput analysis for random workloads of mixed nature for 512KB I/Os.

for these workloads. So far, NVMe/RDMA-56Gbps has better performance than NVMe/RoCE-100Gbps for various workloads and I/O sizes.

5.7 Application-level Evaluation with HDF5

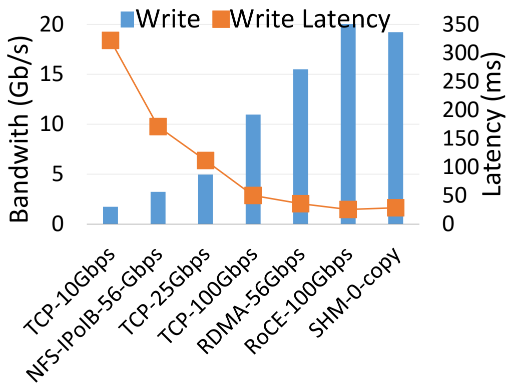
5.7.1 h5bench. HDF5 I/O library is heavily used in a range of scientific applications in high performance computing environments. It is among the top five libraries loaded by applications at the National Energy Research Scientific Computing Center (NERSC) and the Oak Ridge Leadership Computing Facility (OLCF) [18]. Now we demonstrate the performance improvement AF delivers to HDF5 applications over NFS by co-designing h5bench (a popular benchmark that contains representative HDF5 I/O kernels) with NVMe-oAF. Using HDF5 Virtual Object Layer (VOL), we are able to intercept HDF5 APIs and utilize NVMe-oAF's Connection Manager, Locality Awareness, and Buffer Manager components for data storage in the NVMe-SSD.

To evaluate the performance of our adaptive fabric, we run h5bench's read and write I/O kernels from the client. We configure the write kernel to write a 1-dimensional (1D) array of basic datatypes that follows a contiguous pattern both in memory and file layout. We perform write operations with two different configurations to extensively evaluate our design when applications store different number of datasets. The first configuration, called config-1, writes 16*1024*1024 (16M) particles for one 1D array stored as one HDF5 dataset. While, the second configuration, config-2, writes 8*1024*1024 (8M) particles for 8 1D arrays stored as 8 HDF5 datasets. The h5bench reads are similarly configured to perform a full read of the datasets written previously by the write kernel.

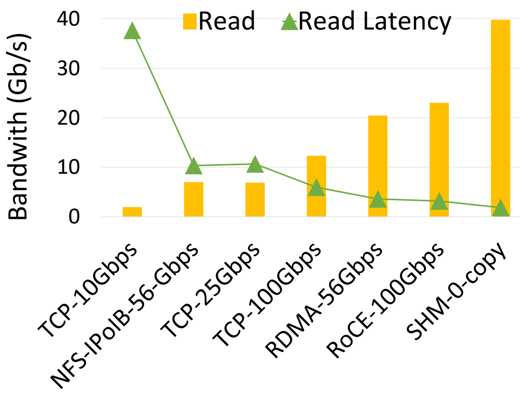
When only one dataset is written to/read from the remote SSD, NVMe-oAF is able to achieve 5.95x higher write bandwidth and 5.68x higher read bandwidth when compared to NFS. The high bandwidth achieved by NVMe-oAF can be atrributed to the low latency of AF as seen in Figures 16. On the contrary, when 8 datasets are written to/read from remote SSD, NVMe-oAF (SHM-0-copy) is 0.53x lower in write bandwidth and 0.41x lower in read bandwidth when compared to NFS. NFS can perform better here due to two reasons. The first reason is the buffering of I/O operations due to the async mount type of NFS and the second one is due to the queuing delay incurred by large-sized I/Os on the SSD. Hence, to extract maximum performance out of NVMe-oAF, we further optimize it to coalesce the I/Os in an application agnostic manner. We then observe that zero-copy along with I/O coalescing helps NVMe-oAF achieve up to 6x and 7x bandwidth improvement for h5bench's write and read operations, respectively, as shown in Figure 17.

5.7.2 Scale-out workloads. Next we evaluate an important use-case where application I/O pattern shifts from remote I/Os to partially remote I/Os (intra-node), which is closer to real-world scenarios. Here, remote I/O occurs over TCP/IP whereas partially remote I/O occurs over shared memory channel. We measure this performance in two different cases - *case-1*, where four clients in one node talk to four remote SSDs located in different physical nodes running inside VMs, and *case-2*, where four clients talk to four remote SSDs located in the same node inside a VM similar to Section 3.1 and later scaled to four nodes. In both these settings h5bench I/O kernels act as clients and each I/O kernel is configured similar to config-1 in Section 5.7.1 to store 16M particles in one HDF5 dataset. Here the legend 'SHM (25%)' indicates that one client out of four uses the shared memory channel whereas the remaining clients use TCP-25Gbps for communication. In Figure 18 we can see that for case-1, SHM (75%) is able to improve the aggregate bandwidth by 1.81x for h5bench write and by 2.98x for h5bench read when compared to SHM (0%). For case-1, we do not report SHM (100%) as it is equivalent to its counterpart in the case-2 setting.

Figure 19 also demonstrates the overall benefit in I/O performance as the ratio of remote to partially remote I/O changes in case-2. With only 25% of the I/O kernels using the shared memory design (the remaining using TCP-25Gbps), the aggregate bandwidth improves by 37% and 66% for h5bench write and read kernels, respectively. Whereas, when all the applications issue partially remote I/Os (SHM 100%), the performance improves by 2.34x and 4.55x for h5bench write and read kernels, respectively, when compared to the TCP-25Gbps. This shows us the advantage and the impact adaptive fabric would have when numerous applications with different communication patterns issue partially remote storage I/O requests.

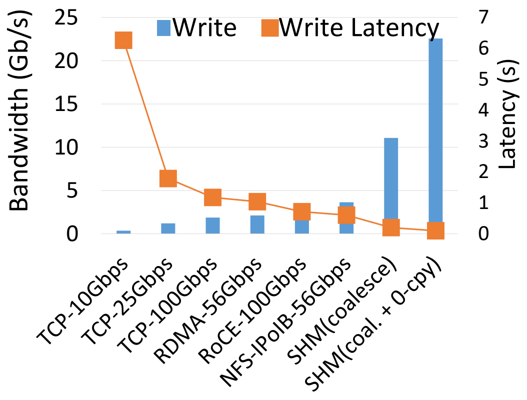


(a) Write

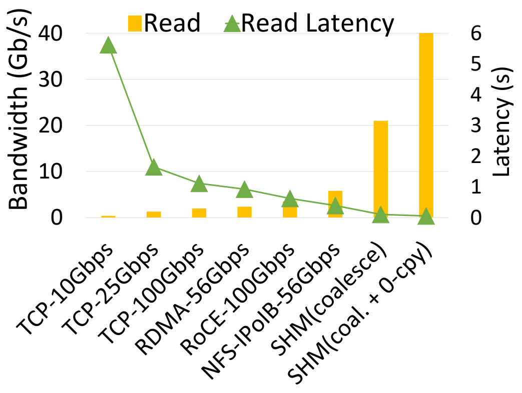


(b) Read

Figure 16: Performance of h5bench I/O kernels for one dataset containing 16 M particles in one timestep.



(a) Write



(b) Read

Figure 17: Performance of h5bench I/O kernels for 8 datasets containing 8 M particles in one timestep.

6 DISCUSSION ON SECURITY

Security is a big aspect in data center scale computing environments because users want a guarantee from platform provider that their data does not get leaked either during communication or application run time. All HPC cloud providers like Amazon Web Services, Microsoft Azure, and Google Cloud allow users to run their applications in VMs. As multiple VMs might share the same physical host/machine, how to protect against malicious applications/VM is an extensive research topic on its own [14, 58, 59, 68]. Co-location with other tenants increases the attack vector due to the presence of covert and side channels [69] and placement vulnerabilities [66] as the malicious entity could break the logical isolation enforced by the virtualization layer [70, 73, 83], deteriorate performance [65, 84], or even worse steal private keys and sensitive data from the application [82].

In our HPC cloud setting, we assume that the physical node/host and the hypervisor/virtualization layer cannot be malicious in nature, i.e., the platform provider would never try to launch an attack on the users applications housed in VMs. The hypervisor is also responsible for initializing and allocating an isolated shared memory channel between the client/user VM and the storage service VM, i.e., hypervisor guarantees that two clients/tenants will never map to the same shared memory. The storage service VM will have multiple shared memory channels assigned to it based on the number of the user VMs on that physical host that need remote storage access. As storage service is managed by the cloud provider, we also assume that the storage service VMs would never try to sniff on the shared memory channels allocated to other clients. Furthermore, we presume that a malicious VM cannot forge a client's identify and pretend as a legitimate application to the storage service in order to gain access of the previously stored data in the SSDs.

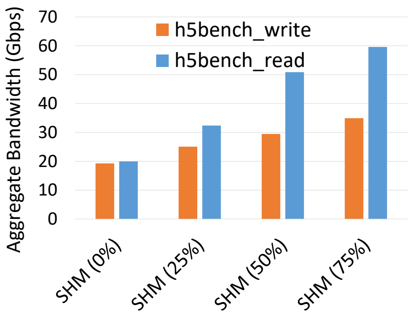


Figure 18: Aggr. bandwidth for h5bench I/O in case-1 (4 nodes).

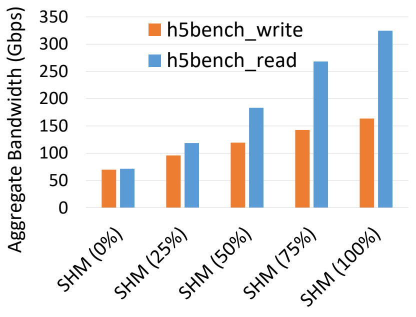


Figure 19: Aggr. bandwidth for h5bench I/O in case-2 (4 nodes).

As our current security model relies on the trustworthiness of the platform and storage service offered, we can beef up the security of shared memory channels by encrypting it with client's private key. This ensures that if a malicious entity does get access to the shared memory channel between the client and the storage service VMs, they would be unsuccessful in tampering and stealing any data. We can further improve security for applications running in HPC clouds by deploying techniques like [60, 77, 81], which detect and hinder any side channel attacks by co-resident malicious VMs. In our future work, we wish to relax some assumptions of our security model and implement techniques that allow HPC applications running on VMs to defend against side-channel and performance degradation attacks in the cloud.

7 RELATED WORK

Some work exists to evaluate the performance of disaggregated storage over NVMe-oF [28, 29, 36, 38]. Klimovic et al. [45, 46] devise a remote flash storage system on top of Ethernet and discuss storage disaggregation architecture in [44]. Kashyap et al. [38] conduct a performance characterization of different NVMe-oF networking protocols and Jia et al. [36] study the performance implications of NVMe-oF on ARM SoCs and Xu et al. [71] evaluate local and remote I/O performance for applications running in containers. Guz et al. [28, 29] compare kernel space NVMe-oF target with iSCSI protocol. NVMe-over-RPMsg [78] emulates remote storage system as local NVMe device for multi-core SoCs to eliminate the long I/O latency of existing NVMe-SSD emulators. The authors use remote processor messaging between guest and remote OS for delivery of read/write commands. NVMe-oAF differs from these works as it focuses on applications directly accessing SSDs using userspace NVMe-oF protocol and proposes a new adaptive fabric that is not designed for a particular HPC workload or scenario.

Some literature aims at increasing the efficiency of accessing local NVMe drives and provides APIs for applications developers. NVMeDirect [42] is a userspace framework that not only allows fast access to NVMe-SSDs but also allows users to define I/O policies like scheduling and caching. KV SSD [7] provides a direct key-value interface to a block device and contains drivers, software packages, and APIs to allow users to benchmark and analyze multiple applications like RocksDB [19] and direct key-value stack's on SSDs.

Inter-VM Shared Memory (IVSHMEM) [6] has been promising in the HPC domain. Zhang et al. [75] present a comprehensive performance evaluation about the benefits of using IVSHMEM for MPI applications in intra-host inter-VM environment. Recently, secure IVHSMEM [64] has been proposed which can be used to further enhance the security between co-located applications using shared memory. Ivanovic et al. [35] show the performance evaluation of IVSHMEM for HPC by integrating their shared memory design to MPICH MPI library. Pickartz et al. [57] investigate the benefits of IVSHMEM during VM migrations and present its comprehensive performance evaluation for intra-host VMs. C-GDR [74] propose locality-aware and container-aware designs to alleviate communication bottlenecks on GPU-enabled clouds.

8 CONCLUSION

In this paper, we propose a novel concept, called Adaptive Fabric, for accelerating NVMe-oF protocols. The key idea of our work is to adaptively and transparently leverage and optimize both commonly available I/O paths like TCP/IP and shared memory in the HPC cloud architecture. Based on this, we propose designs and optimizations for NVMe-over-Adaptive-Fabric (i.e., NVMe-oAF) protocols. We further co-design our proposed NVMe-oAF with Intel SPDK and HDF5 storage runtimes. Extensive evaluations demonstrate that our design is able to achieve up to 7.1x bandwidth increase and up to 4.2x latency reduction for various workloads compared to conventional NVMe-oF protocols over high-speed Ethernet networks (e.g., 10/25/100 Gbps). For HDF5 workloads, NVMe-oAF attains up to 7x bandwidth improvement over NFS. In the future, we will try to make our designs support more communication optimizations with RDMA or user-level TCP/IP, and make our designs open-source available.

9 ACKNOWLEDGEMENT

We would like to sincerely thank Shashank Gugnani for initial discussions and Yuke Li for help in conducting some of the experiments. We would like to thank Suren Byna from Lawrence Berkeley National Laboratory for his suggestions on benchmarking with h5bench. We want to thank the anonymous reviewers for their insightful comments and suggestions. For compute resources, we would like to thank CloudLab and Chameleon Cloud. This work was supported in part by the NSF research grant CCF #2132049 and a COR grant from University of California, Merced.

REFERENCES

[1] 2006. The HDF Group. http://www.hdfgroup.org/solutions/hdf5/
[2] 2019. NVM ExpressTM over Fabrics Revision 1.1. https://nvmexpress.org/wp-content/uploads/NVMe-over-Fabrics-1.1-2019.10.22-Ratified.pdf
[3] 2021. Top500. https://www.top500.org/statistics/list/
[4] 2022. Data Plane Development Kit. https://www.dpdk.org/
[5] 2022. Direct Memory Access (DMA) From User Space. https://spdk.io/doc/memory.html
[6] 2022. Inter-VM Shared Memory Device. https://www.qemu.org/docs/master/system/devices/ivshmem.html
[7] 2022. KV SSD. https://github.com/OpenMPDK/KVSSD
[8] 2022. NVM Express. https://nvmexpress.org/
[9] 2022. NVMe-over-Fabrics Specification. https://nvmexpress.org/developers/nvme-of-specification/
[10] 2022. QEMU. https://www.qemu.org/
[11] 2022. Single Root I/O Virtualization. https://pcisig.com/
[12] 2022. SPDK NVMe perf Benchmark. https://github.com/spdk/spdk/tree/master/examples/nvme/perf
[13] 2022. What is SPDK . https://spdk.io/doc/about.html
[14] Mazhar Ali, Samee U. Khan, and Athanasios V. Vasilakos. 2015. Security in Cloud Computing: Opportunities and Challenges. *Information Sciences* 305 (2015), 357–383. https://doi.org/10.1016/j.ins.2015.01.025
[15] Jens Axboe. 2022. Flexible IO Tester (FIO) ver 3.13. https://github.com/axboe/fio
[16] Shajulin Benedict. 2013. Performance Issues and Performance Analysis Tools for HPC Cloud Applications: A Survey. *Computing* 95, 2 (2013), 89–108.
[17] Tim Bisson, Ke Chen, Changho Choi, Vijay Balakrishnan, and Yang-suk Kee. 2018. Crail-KV: A High-Performance Distributed Key-Value Store Leveraging Native KV-SSDs over NVMe-oF. In *2018 IEEE 37th International Performance Computing and Communications Conference (IPCCC)*. 1–8. https://doi.org/10.1109/PCCC.2018.8710776
[18] Suren Byna, M Scot Breitenfeld, Bin Dong, Quincey Koziol, Elena Pourmal, Dana Robinson, Jerome Soumagne, Houjun Tang, Venkatram Vishwanath, and Richard Warren. 2020. ExaHDF5: Delivering Efficient Parallel I/O on Exascale Computing Systems. *Journal of Computer Science and Technology* 35, 1 (2020), 145–160.
[19] Zhichao Cao, Siying Dong, Sagar Vemuri, and David H.C. Du. 2020. Characterizing, Modeling, and Benchmarking RocksDB Key-Value Workloads at Facebook. In *18th USENIX Conference on File and Storage Technologies (FAST 20)*. USENIX Association, Santa Clara, CA, 209–223. https://www.usenix.org/conference/fast20/presentation/cao-zhichao
[20] Adam G. Carlyle, Stephen L. Harrell, and Preston M. Smith. 2010. Cost-Effective HPC: The Community or the Cloud?. In *2010 IEEE Second International Conference on Cloud Computing Technology and Science*. 169–176. https://doi.org/10.1109/CloudCom.2010.115
[21] Feng Chen, Tian Luo, and Xiaodong Zhang. 2011. CAFTL: A Content-Aware Flash Translation Layer Enhancing the Lifespan of Flash Memory based Solid State Drives. In *9th USENIX Conference on File and Storage Technologies (FAST 11)*. USENIX Association, San Jose, CA. https://www.usenix.org/conference/fast11/caftl-content-aware-flash-translation-layer-enhancing-lifespan-flash-memory-based
[22] Dmitry Duplyakin, Robert Ricci, Aleksander Maricq, Gary Wong, Jonathon Duerig, Eric Eide, Leigh Stoller, Mike Hibler, David Johnson, Kirk Webb, Aditya Akella, Kuangching Wang, Glenn Ricart, Larry Landweber, Chip Elliott, Michael Zink, Emmanuel Cecchet, Snigdhaswin Kar, and Prabodh Mishra. 2019. The Design and Operation of CloudLab. In *Proceedings of the USENIX Annual Technical Conference (ATC)*. 1–14. https://www.flux.utah.edu/paper/duplyakin-atc19
[23] Roberto R. Expósito, Guillermo L. Taboada, Sabela Ramos, Juan Touriño, and Ramón Doallo. 2013. Performance Analysis of HPC Applications in the Cloud. *Future Generation Computer Systems* 29, 1 (2013), 218–229. https://doi.org/10.1016/j.future.2012.06.009 Including Special section: AIRCC-NetCoM 2009 and Special section: Clouds and Service-Oriented Architectures.
[24] Sanjay Ghemawat, Howard Gobioff, and Shun-Tak Leung. 2003. The Google File System. In *Proceedings of the Nineteenth ACM Symposium on Operating Systems Principles* (Bolton Landing, NY, USA) *(SOSP '03)*. Association for Computing Machinery, New York, NY, USA, 29–43. https://doi.org/10.1145/945445.945450
[25] Abhishek Gupta, Paolo Faraboschi, Filippo Gioachin, Laxmikant V. Kale, Richard Kaufmann, Bu-Sung Lee, Verdi March, Dejan Milojicic, and Chun Hui Suen. 2016. Evaluating and Improving the Performance and Scheduling of HPC Applications in Cloud. *IEEE Transactions on Cloud Computing* 4, 3 (2016), 307–321. https://doi.org/10.1109/TCC.2014.2339858
[26] Abhishek Gupta, Laxmikant V. Kale, Filippo Gioachin, Verdi March, Chun Hui Suen, Bu-Sung Lee, Paolo Faraboschi, Richard Kaufmann, and Dejan Milojicic. 2013. The Who, What, Why, and How of High Performance Computing in the Cloud. In *2013 IEEE 5th International Conference on Cloud Computing Technology and Science*, Vol. 1. 306–314. https://doi.org/10.1109/CloudCom.2013.47
[27] Abhishek Gupta and Dejan Milojicic. 2011. Evaluation of HPC Applications on Cloud. In *2011 Sixth Open Cirrus Summit*. 22–26. https://doi.org/10.1109/OCS.2011.10
[28] Zvika Guz, Harry (Huan) Li, Anahita Shayesteh, and Vijay Balakrishnan. 2017. NVMe-over-Fabrics Performance Characterization and the Path to Low-Overhead Flash Disaggregation. In *Proceedings of the 10th ACM International Systems and Storage Conference (SYSTOR'17)*. Article 16, 9 pages.
[29] Zvika Guz, Harry (Huan) Li, Anahita Shayesteh, and Vijay Balakrishnan. 2018. Performance Characterization of NVMe-over-Fabrics Storage Disaggregation. *ACM Trans. Storage* 14, 4, Article 31 (Dec. 2018), 18 pages. https://doi.org/10.1145/3239563
[30] Mingzhe Hao, Gokul Soundararajan, Deepak Kenchammana-Hosekote, Andrew A. Chien, and Haryadi S. Gunawi. 2016. The Tail at Store: A Revelation from Millions of Hours of Disk and SSD Deployments. In *14th USENIX Conference on File and Storage Technologies (FAST 16)*. USENIX Association, Santa Clara, CA, 263–276. https://www.usenix.org/conference/fast16/technical-sessions/presentation/hao
[31] Jahanzeb Maqbool Hashmi, Shulei Xu, Bharath Ramesh, Mohammadreza Bayatpour, Hari Subramoni, and Dhabaleswar K. DK Panda. 2020. Machine-agnostic and Communication-aware Designs for MPI on Emerging Architectures. In *2020 IEEE International Parallel and Distributed Processing Symposium (IPDPS)*. 32–41. https://doi.org/10.1109/IPDPS47924.2020.00014
[32] Zhiqiang He, Dongyang Wang, Binzhang Fu, Kun Tan, Bei Hua, Zhi-Li Zhang, and Kai Zheng. 2020. MasQ: RDMA for Virtual Private Cloud. 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, 1–14. https://doi.org/10.1145/3387514.3405849
[33] Jaehyun Hwang, Qizhe Cai, Ao Tang, and Rachit Agarwal. 2020. TCP ≈ RDMA: CPU-efficient Remote Storage Access with i10 . In *17th USENIX Symposium on Networked Systems Design and Implementation (NSDI 20)*. USENIX Association, Santa Clara, CA, 127–140. https://www.usenix.org/conference/nsdi20/presentation/hwang
[34] Jaehyun Hwang, Midhul Vuppalapati, Simon Peter, and Rachit Agarwal. 2021. Rearchitecting Linux Storage Stack for µs Latency and High Throughput. In *15th USENIX Symposium on Operating Systems Design and Implementation (OSDI 21)*. USENIX Association, 113–128. https://www.usenix.org/conference/osdi21/presentation/hwang
[35] Pavle Ivanovic and Harald Richter. 2018. Performance Analysis of Ivshmem for High-Performance Computing in Virtual Machines. *Journal of Physics: Conference Series* 960 (jan 2018), 012015. https://doi.org/10.1088/1742-6596/960/1/012015
[36] Yichen Jia, Eric Anger, and Feng Chen. 2019. When NVMe over Fabrics Meets Arm: Performance and Implications. In *2019 35th Symposium on Mass Storage Systems and Technologies (MSST)*. 134–140.
[37] Abhijeet Joglekar, Michael E. Kounavis, and Frank L. Berry. 2005. A Scalable and High Performance Software iSCSI Implementation. In *Proceedings of the 4th Conference on USENIX Conference on File and Storage Technologies - Volume 4 (FAST'05)*. USENIX Association, USA, 20.
[38] Arjun Kashyap, Shashank Gugnani, and Xiaoyi Lu. 2021. Impact of Commodity Networks on Storage Disaggregation with NVMe-oF. In *Benchmarking, Measuring, and Optimizing*, Felix Wolf and Wanling Gao (Eds.). Springer International Publishing, Cham, 41–56.
[39] Kate Keahey, Jason Anderson, Zhuo Zhen, Pierre Riteau, Paul Ruth, Dan Stanzione, Mert Cevik, Jacob Colleran, Haryadi S. Gunawi, Cody Hammock, Joe Mambretti, Alexander Barnes, François Halbach, Alex Rocha, and Joe Stubbs. 2020. Lessons Learned from the Chameleon Testbed. In *Proceedings of the 2020 USENIX Annual Technical Conference (USENIX ATC '20)*. USENIX Association.
[40] H. M. Khosravi, Abhijeet Joglekar, and Ravi Iyer. 2005. Performance Characterization of iSCSI Processing in a Server Platform. In *PCCC 2005. 24th IEEE International Performance, Computing, and Communications Conference, 2005.* 99–107.
[41] Daehyeok Kim, Tianlong Yu, Hongqiang Harry Liu, Yibo Zhu, Jitu Padhye, Shachar Raindel, Chuanxiong Guo, Vyas Sekar, and Srinivasan Seshan. 2019. FreeFlow: Software-based Virtual RDMA Networking for Containerized Clouds. In *16th USENIX Symposium on Networked Systems Design and Implementation (NSDI 19)*. USENIX Association, Boston, MA, 113–126. https://www.usenix.org/conference/nsdi19/presentation/kim
[42] Hyeong-Jun Kim, Young-Sik Lee, and Jin-Soo Kim. 2016. NVMeDirect: A User-space I/O Framework for Application-specific Optimization on NVMe SSDs. In *8th USENIX Workshop on Hot Topics in Storage and File Systems (HotStorage 16)*. USENIX Association, Denver, CO. https://www.usenix.org/conference/hotstorage16/workshop-program/presentation/kim
[43] Jaeho Kim, Donghee Lee, and Sam H. Noh. 2015. Towards SLO Complying SSDs Through OPS Isolation. In *13th USENIX Conference on File and Storage Technologies (FAST 15)*. USENIX Association, Santa Clara, CA, 183–189. https://www.usenix.org/conference/fast15/technical-sessions/presentation/kim_jaeho
[44] Ana Klimovic, Christos Kozyrakis, Eno Thereska, Binu John, and Sanjeev Kumar. 2016. Flash Storage Disaggregation. In *Proceedings of the Eleventh European Conference on Computer Systems* (London, United Kingdom) *(EuroSys '16)*. Association for Computing Machinery, New York, NY, USA, Article 29, 15 pages. https://doi.org/10.1145/2901318.2901337

[45] Ana Klimovic, Heiner Litz, and Christos Kozyrakis. 2017. ReFlex: Remote Flash ≈ Local Flash. In *Proceedings of the Twenty-Second International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS '17)*. 345–359.

[46] Ana Klimovic, Yawen Wang, Patrick Stuedi, Animesh Trivedi, Jonas Pfefferle, and Christos Kozyrakis. 2018. Pocket: Elastic Ephemeral Storage for Serverless Analytics. In *Proceedings of the 13th USENIX Conference on Operating Systems Design and Implementation (OSDI'18)*. USENIX Association, USA, 427–444.

[47] Tonglin Li, Suren Byna, Quincey Koziol, Houjun Tang, Jean Luca Bez, and Qiao Kang. 2021. h5bench: HDF5 I/O Kernel Suite for Exercising HPC I/O Patterns. In *Proceedings of Cray User Group Meeting, CUG 2021*.

[48] Tianxi Li, Haiyang Shi, and Xiaoyi Lu. 2021. HatRPC: Hint-Accelerated Thrift RPC over RDMA. In *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis* (St. Louis, Missouri) *(SC '21)*. Association for Computing Machinery, New York, NY, USA, Article 36, 14 pages. https://doi.org/10.1145/3458817.3476191

[49] Liran Liss. 2017. The Linux SoftRoce Driver. In *OpenFabrics Annual Workshop*.

[50] Renping Liu, Xianzhang Chen, Yujuan Tan, Runyu Zhang, Liang Liang, and Duo Liu. 2020. SSDKeeper: Self-Adapting Channel Allocation to Improve the Performance of SSD Devices. In *2020 IEEE International Parallel and Distributed Processing Symposium (IPDPS)*. 966–975. https://doi.org/10.1109/IPDPS47924.2020.00103

[51] Youyou Lu, Jiwu Shu, and Weimin Zheng. 2013. Extending the Lifetime of Flash-based Storage through Reducing Write Amplification from File Systems. In *11th USENIX Conference on File and Storage Technologies (FAST 13)*. USENIX Association, San Jose, CA, 257–270. https://www.usenix.org/conference/fast13/technical-sessions/presentation/lu_youyou

[52] Aniruddha Marathe, Rachel Harris, David K. Lowenthal, Bronis R. de Supinski, Barry Rountree, Martin Schulz, and Xin Yuan. 2013. A Comparative Study of High-Performance Computing on the Cloud. In *Proceedings of the 22nd International Symposium on High-Performance Parallel and Distributed Computing* (New York, New York, USA) *(HPDC '13)*. Association for Computing Machinery, New York, NY, USA, 239–250. https://doi.org/10.1145/2493123.2462919

[53] Jaehong Min, Ming Liu, Tapan Chugh, Chenxingyu Zhao, Andrew Wei, In Hwan Doh, and Arvind Krishnamurthy. 2021. Gimbal: Enabling Multi-Tenant Storage Disaggregation on SmartNIC JBOFs. In *Proceedings of the 2021 ACM SIGCOMM 2021 Conference* (Virtual Event, USA) *(SIGCOMM '21)*. Association for Computing Machinery, New York, NY, USA, 106–122. https://doi.org/10.1145/3452296.3472940

[54] Marco A. S. Netto, Rodrigo N. Calheiros, Eduardo R. Rodrigues, Renato L. F. Cunha, and Rajkumar Buyya. 2018. HPC Cloud for Scientific and Business Applications: Taxonomy, Vision, and Research Challenges. *ACM Comput. Surv.* 51, 1, Article 8 (Jan 2018), 29 pages. https://doi.org/10.1145/3150224

[55] Simon Ostermann, Alexandria Iosup, Nezih Yigitbasi, Radu Prodan, Thomas Fahringer, and Dick Epema. 2010. A Performance Analysis of EC2 Cloud Computing Services for Scientific Computing. In *Cloud Computing*, Dimiter R. Avresky, Michel Diaz, Arndt Bode, Bruno Ciciani, and Eliezer Dekel (Eds.). Springer Berlin Heidelberg, Berlin, Heidelberg, 115–131.

[56] Jonas Pfefferle, Patrick Stuedi, Animesh Trivedi, Bernard Metzler, Ionnis Koltsidas, and Thomas R. Gross. 2015. A Hybrid I/O Virtualization Framework for RDMA-Capable Network Interfaces. In *Proceedings of the 11th ACM SIGPLAN/SIGOPS International Conference on Virtual Execution Environments* (Istanbul, Turkey) *(VEE '15)*. Association for Computing Machinery, New York, NY, USA, 17–30. https://doi.org/10.1145/2731186.2731200

[57] Simon Pickartz, Jonas Baude, Stefan Lankes, and Antonello Monti. 2017. A Locality-Aware Communication Layer for Virtualized Clusters. In *High Performance Computing*, Julian M. Kunkel, Rio Yokota, Michela Taufer, and John Shalf (Eds.). Springer International Publishing, Cham, 605–616.

[58] Gururaj Ramachandra, Mohsin Iftikhar, and Farrukh Aslam Khan. 2017. A Comprehensive Survey on Security in Cloud Computing. *Procedia Computer Science* 110 (2017), 465–472. https://doi.org/10.1016/j.procs.2017.06.124 14th International Conference on Mobile Systems and Pervasive Computing (MobiSPC 2017) / 12th International Conference on Future Networks and Communications (FNC 2017) / Affiliated Workshops.

[59] S Ramgovind, M M Eloff, and E Smith. 2010. The Management of Security in Cloud Computing. In *2010 Information Security for South Africa*. 1–7. https://doi.org/10.1109/ISSA.2010.5588290

[60] U.K. Muhammed Sadique and Divya James. 2016. A Novel Approach to Prevent Cache-based Side-Channel Attack in the Cloud. *Procedia Technology* 25 (2016), 232–239. https://doi.org/10.1016/j.protcy.2016.08.102 1st Global Colloquium on Recent Advancements and Effectual Researches in Engineering, Science and Technology - RAEREST 2016 on April 22nd 23rd April 2016.

[61] Deboleena Sakalley. 2017. Using FPGAs to accelerate NVMe-of based Storage Networks. In *Flash Memory Summit 2017*. 8–11.

[62] Russel Sandberg, David Goldberg, Steve Kleiman, Dan Walsh, and Bob Lyon. 1985. Design and Implementation of the Sun Network Filesystem. In *Proceedings of the Summer USENIX Conference*. 119–130.

[63] Konstantin Shvachko, Hairong Kuang, Sanjay Radia, and Robert Chansler. 2010. The Hadoop Distributed File System. In *2010 IEEE 26th Symposium on Mass Storage Systems and Technologies (MSST)*. 1–10. https://doi.org/10.1109/MSST.2010.5496972

[64] Shesha Sreenivasamurthy and Ethan Miller. 2019. Sivshm: Secure Inter-VM Shared Memory. *arXiv preprint arXiv:1909.10377* (2019).

[65] Venkatanathan Varadarajan, Thawan Kooburat, Benjamin Farley, Thomas Ristenpart, and Michael M. Swift. 2012. Resource-Freeing Attacks: Improve Your Cloud Performance (at Your Neighbor's Expense). In *Proceedings of the 2012 ACM Conference on Computer and Communications Security* (Raleigh, North Carolina, USA) *(CCS '12)*. Association for Computing Machinery, New York, NY, USA, 281–292. https://doi.org/10.1145/2382196.2382228

[66] Venkatanathan Varadarajan, Yinqian Zhang, Thomas Ristenpart, and Michael Swift. 2015. A Placement Vulnerability Study in Multi-Tenant Public Clouds. In *24th USENIX Security Symposium (USENIX Security 15)*. USENIX Association, Washington, D.C., 913–928. https://www.usenix.org/conference/usenixsecurity15/technical-sessions/presentation/varadarajan

[67] Dongyang Wang, Binzhang Fu, Gang Lu, Kun Tan, and Bei Hua. 2019. VSocket: Virtual Socket Interface for RDMA in Public Clouds. In *Proceedings of the 15th ACM SIGPLAN/SIGOPS International Conference on Virtual Execution Environments* (Providence, RI, USA) *(VEE 2019)*. Association for Computing Machinery, New York, NY, USA, 179–192. https://doi.org/10.1145/3313808.3313813

[68] Qian Wang, Cong Wang, Kui Ren, Wenjing Lou, and Jin Li. 2011. Enabling Public Auditability and Data Dynamics for Storage Security in Cloud Computing. *IEEE Transactions on Parallel and Distributed Systems* 22, 5 (2011), 847–859. https://doi.org/10.1109/TPDS.2010.183

[69] Zhenghong Wang and Ruby B. Lee. 2006. Covert and Side Channels Due to Processor Architecture. In *2006 22nd Annual Computer Security Applications Conference (ACSAC'06)*. 473–482. https://doi.org/10.1109/ACSAC.2006.20

[70] Zhenyu Wu, Zhang Xu, and Haining Wang. 2012. Whispers in the Hyper-space: High-speed Covert Channel Attacks in the Cloud. In *21st USENIX Security Symposium (USENIX Security 12)*. USENIX Association, Bellevue, WA, 159–173. https://www.usenix.org/conference/usenixsecurity12/technical-sessions/presentation/wu

[71] Qiumin Xu, Manu Awasthi, Krishna Malladi, Janki Bhimani, Jingpei Yang, Murali Annavaram, and Ming Hsieh. 2017. Performance Analysis of Containerized Applications on Local and Remote Storage. In *International Conference on Massive Storage Systems and Technology*.

[72] Ziye Yang, James R. Harris, Benjamin Walker, Daniel Verkamp, Changpeng Liu, Cunyin Chang, Gang Cao, Jonathan Stern, Vishal Verma, and Luse E. Paul. 2017. SPDK: A Development Kit to Build High Performance Storage Applications. In *2017 IEEE International Conference on Cloud Computing Technology and Science (CloudCom)*. 154–161.

[73] Yuval Yarom and Katrina Falkner. 2014. FLUSH+RELOAD: A High Resolution, Low Noise, L3 Cache Side-Channel Attack. In *23rd USENIX Security Symposium (USENIX Security 14)*. USENIX Association, San Diego, CA, 719–732. https://www.usenix.org/conference/usenixsecurity14/technical-sessions/presentation/yarom

[74] Jie Zhang, Xiaoyi Lu, Ching-Hsiang Chu, and Dhabaleswar K. Panda. 2019. C-GDR: High-Performance Container-Aware GPUDirect MPI Communication Schemes on RDMA Networks. In *2019 IEEE International Parallel and Distributed Processing Symposium (IPDPS)*. 242–251. https://doi.org/10.1109/IPDPS.2019.00034

[75] Jie Zhang, Xiaoyi Lu, Jithin Jose, Rong Shi, and Dhabaleswar K. (DK) Panda. 2014. Can Inter-VM Shmem Benefit MPI Applications on SR-IOV Based Virtualized Infiniband Clusters?. In *Euro-Par 2014 Parallel Processing*, Fernando Silva, Inês Dutra, and Vítor Santos Costa (Eds.). Springer International Publishing, Cham, 342–353.

[76] Jie Zhang, Xiaoyi Lu, and Dhabaleswar K. Panda. 2016. High Performance MPI Library for Container-Based HPC Cloud on InfiniBand Clusters. In *2016 45th International Conference on Parallel Processing (ICPP)*. 268–277. https://doi.org/10.1109/ICPP.2016.38

[77] Tianwei Zhang, Yinqian Zhang, and Ruby B. Lee. 2016. CloudRadar: A Real-Time Side-Channel Attack Detection System in Clouds. In *Research in Attacks, Intrusions, and Defenses*, Fabian Monrose, Marc Dacier, Gregory Blanc, and Joaquin Garcia-Alfaro (Eds.). Springer International Publishing, Cham, 118–140.

[78] Xiaohao Zhang, Yunjie Li, and Gang Chen. 2020. NVMe-over-RPMsg: A Virtual Storage Device Model Applied to Heterogeneous Multi-Core SoCs. In *2020 10th Annual Computing and Communication Workshop and Conference (CCWC)*. 0821–0826. https://doi.org/10.1109/CCWC47524.2020.9031144

[79] Xiaoyi Zhang, Feng Zhu, Shu Li, Kun Wang, Wei Xu, and Dengcai Xu. 2021. Optimizing Performance for Open-Channel SSDs in Cloud Storage System. In *2021 IEEE International Parallel and Distributed Processing Symposium (IPDPS)*. 902–911. https://doi.org/10.1109/IPDPS49936.2021.00099

[80] Xiaoyi Zhang, Feng Zhu, Shu Li, Kun Wang, Wei Xu, and Dengcai Xu. 2021. Optimizing Performance for Open-Channel SSDs in Cloud Storage System. In *2021 IEEE International Parallel and Distributed Processing Symposium (IPDPS)*. 902–911. https://doi.org/10.1109/IPDPS49936.2021.00099

[81] Yinqian Zhang, Ari Juels, Alina Oprea, and Michael K. Reiter. 2011. HomeAlone: Co-residency Detection in the Cloud via Side-Channel Analysis. In *2011 IEEE Symposium on Security and Privacy*. 313–328. https://doi.org/10.1109/SP.2011.31

[82] Yinqian Zhang, Ari Juels, Michael K. Reiter, and Thomas Ristenpart. 2012. Cross-VM Side Channels and Their Use to Extract Private Keys. In *Proceedings of the 2012 ACM Conference on Computer and Communications Security* (Raleigh, North Carolina, USA) *(CCS '12)*. Association for Computing Machinery, New York, NY, USA, 305–316. https://doi.org/10.1145/2382196.2382230

[83] Yinqian Zhang, Ari Juels, Michael K. Reiter, and Thomas Ristenpart. 2014. Cross-Tenant Side-Channel Attacks in PaaS Clouds. In *Proceedings of the 2014 ACM SIGSAC Conference on Computer and Communications Security* (Scottsdale, Arizona, USA) *(CCS '14)*. Association for Computing Machinery, New York, NY, USA, 990–1003. https://doi.org/10.1145/2660267.2660356

[84] Fangfei Zhou, Manish Goel, Peter Desnoyers, and Ravi Sundaram. 2013. Scheduler Vulnerabilities and Coordinated Attacks in Cloud Computing. *Journal of Computer Security* 21, 4 (2013), 533–559.

[85] Yue Zhu, Weikuan Yu, Bing Jiao, Kathryn Mohror, Adam Moody, and Fahim Chowdhury. 2019. Efficient User-Level Storage Disaggregation for Deep Learning. In *2019 IEEE International Conference on Cluster Computing (CLUSTER)*. 1–12. https://doi.org/10.1109/CLUSTER.2019.8891023