2024 IEEE International Parallel and Distributed Processing Symposium (IPDPS) | 979-8-3503-8711-7/24/$31.00 ©2024 IEEE | DOI: 10.1109/IPDPS57955.2024.00052

# NVMe-oPF: Designing Efficient Priority Schemes for NVMe-over-Fabrics with Multi-Tenancy Support

Darren Ng†, Andrew Lin†, Arjun Kashyap†, Guanpeng Li‡, and Xiaoyi Lu†*
†University of California, Merced, Merced, CA, USA
{dng350, alin85, akashyap5, xiaoyi.lu}@ucmerced.edu
‡University of Iowa, Iowa City, IA, USA
guanpeng-li@uiowa.edu

***Abstract*—Resource disaggregation is prevalent in datacenters since it provides high resource utilization when compared to servers dedicated to either compute, memory, or storage. NVMe-over-Fabrics (NVMe-oF) is the standardized protocol for accessing disaggregated network storage. Currently, the NVMe-oF specification lacks semantics to prioritize I/O requests based on different application needs. Since applications have varying goals — latency-sensitive or throughput-critical I/O — we need to design efficient schemes to allow applications to specify the type of performance they wish to achieve. To this end, we propose a new NVMe-over-Priority-Fabrics (NVMe-oPF) protocol with multi-tenancy support that allows applications to specify whether to optimize for latency or throughput. NVMe-oPF proposes coalescing request completions, lock-free optimization, zero-copy queues, out-of-order request completion handling, and window size optimization for the specific I/O patterns, queue depths, and I/O sizes that yield the best performance. Our NVMe-oPF-10 Gbps can achieve up to 2.94X improvement in throughput and reduces tail latency by up to 32.1% for highly concurrent multi-tenant read workloads when compared to the state-of-the-art userspace NVMe-oF runtime design in Intel Storage Performance Development Kit (SPDK). For write workloads with 100 Gbps, NVMe-oPF achieves a 32.6% increase in throughput while maintaining low latency compared to SPDK. We also bring performance benefits to the application level with HDF5 by increasing write workload throughput by 25.2% in larger-scale experiments.**

***Index Terms*—Disaggregated Storage, Multi-Tenancy, NVMe-over-Fabric (NVMe-oF), Priority Scheme**

## I. Introduction

Traditional High-Performance Computing (HPC) and cloud applications have been running on servers that house all resources, e.g., compute, memory, and storage, within a single server box. However, the paradigm has shifted with the advent of resource disaggregation [1]–[10]. HPC and cloud providers are now witnessing heightened resource utilization, a trend that translates to reduced infrastructure costs. Disaggregating resources from a physical server offers two pivotal advantages. Firstly, providers can pool and scale compute, memory, and storage independently without being limited by the capacity of a single physical server [2], [3], [7], [11]. Secondly, and notably, the focus of this paper, is the facilitation of multi-tenancy, where multiple diverse user applications can seamlessly share underlying hardware in an application-agnostic manner [11]–[13].

* is the corresponding author.

### A. Motivation

Despite being a highly sought-after feature in data center and HPC environments, implementing multi-tenancy presents notable challenges due to the diverse performance requirements of different applications. Existing storage infrastructures, underpinned by underlying disaggregated storage runtimes like NVMe-oF [14], face a substantial predicament. While these runtimes permit requests from various applications, they lack the capability to tailor performance to each application's unique demands. Figure 1 visually depicts this challenge on how applications with distinct priorities interact with a storage service, exposing NVMe SSDs across different networks using NVMe-oF.

On the other hand, as per its design standard, the NVMe-oF protocol permits multiple concurrent tenants to access the storage device. However, the NVMe-oF runtime does not distinguish the specific requirements of each application. This uniform treatment results in significant overheads for each tenant connected to the NVMe SSD. To address this issue, our approach defines multi-tenancy in a disaggregated environment as follows: each application attains a specified performance optimization strategy that is independent of other concurrent applications. However, achieving this multi-tenant-aware disaggregated storage runtime presents several challenges. Firstly, it necessitates the provision of a throughput-optimized flow tailored for applications with specified throughput requirements. Secondly, it resorts to bypassing throughput-intensive computation queues for applications dependent on low latency. Lastly, it demands efficient maintenance and management of interconnected communication between all tenants and the target storage. This scenario prompts a fundamental question: *What aspects within the current NVMe-oF specification lack support for multi-tenancy?*

### B. Challenges

Upon examination, we identified three major factors that need addressing in the current NVMe-oF runtime: computation order, semantic data passing, and priority/multi-tenant management. We subsequently formulate specific research questions for each factor, paving the way for further exploration and innovation within multi-tenant-aware storage runtimes.

**Computation Order:** *How should NVMe-oF runtimes handle the computation orderings to support diverse optimization methods?*

1530-2075/24/$31.00 ©2024 IEEE
DOI 10.1109/IPDPS57955.2024.00052

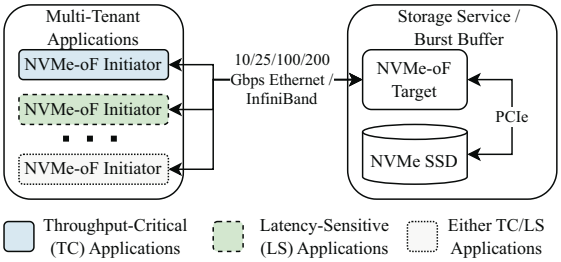



Fig. 1: Multi-Tenancy Requirements of NVMe-oF Architecture on Priority Schemes.

Disaggregated storage typically processes requests in a first-in-first-out (FIFO) manner. Consequently, an application prioritizing low latency might find itself delayed by a backlog of requests from a high-throughput application, not satisfying the multi-tenancy requirement. Moreover, latency-sensitive requests could disrupt throughput-critical requests, forcing the target to switch between processing different types of requests within a single batch. This situation adversely affects the throughput-critical requests of high-throughput applications, thus failing to meet multi-tenancy requirements.

**Semantic Data Passing:** *What type of data must be included within requests to inform the NVMe-oF runtime about both priority and tenant structure across the network?* Current disaggregated storage protocols, such as NVMe-oF, lack support for priority schemes enabling the storage system to differentiate I/O requests. Consequently, remote storage runtimes cannot interpret or manage varied requests to optimize specific performance goals as desired by the applications. This capability hinges on embedding semantic data within each request, indicating the application's optimization strategy. In addition, the NVMe-oF runtime also needs awareness of the number of connected tenants and which ones require what optimizations to align with multi-tenancy objectives. Towards this, semantic data passing becomes imperative.

**Multi-Tenant Management:** *Where do multi-tenant aware systems reside in a disaggregated setting, and how can they manage high-congestion request flow?* In multi-tenant scenarios, where congestion can significantly escalate, each host must be multi-tenant aware to facilitate the efficient handling of semantic data and computation orders. This means that the disaggregated storage runtime must recognize which request was sent by which client and subsequently handle each client's request according to their specific optimization needs. As a result, multi-tenant-aware libraries cannot be confined to a single machine.

### C. Contributions

In response to these questions, this paper introduces the NVMe-over-Priority-Fabrics (NVMe-oPF) protocol, which empowers applications to specify whether they prioritize latency or throughput. Our priority schemes enhance the NVMe-oF target to enable multi-tenant awareness. We implement one Priority Manager (PM) per NVMe-oPF runtime (distinct PMs for NVMe-oPF initiator and target), enabling effective computation order and semantic data passing for seamless multi-tenant communication. Our focus on userspace NVMe-oF runtime is deliberate, as it sidesteps context-switching overhead and ensures low-latency access to the underlying storage device.

NVMe-oPF aims to demonstrate the performance enhancements achieved by conveying semantic information from workloads. For instance, it discerns whether I/O requests are latency-sensitive or throughput-critical, enabling the NVMe-oF runtime to optimize accordingly. To enhance the performance of throughput-critical I/O requests, we employ a coalescing strategy that consolidates multiple request completion notifications into one. Concurrently, for applications that do not prioritize throughput, NVMe-oPF allows them to designate specific I/O requests over fabric as latency-sensitive. This tagging mechanism ensures the prioritization of latency-sensitive requests with concurrent throughput-critical requests. Through these innovations, the proposed NVMe-oPF runtime can adeptly handle diverse application objectives from multiple concurrent applications, providing robust multi-tenancy support in disaggregated storage environments.

To implement NVMe-oPF, we have chosen to enhance Intel SPDK with priority schemes and multi-tenant support owing to its established reputation as a userspace storage library in HPC systems. For instance, Intel Distributed Asynchronous Object Storage (DAOS), a widely acclaimed exascale storage stack [15]–[17], relies on SPDK [18] internally. However, traditional NVMe-oF storage runtimes struggle to interpret or manage varied requests, hindering the optimization of specific performance goals desired by applications. To bridge this gap, we introduce priority schemes into the SPDK-based NVMe-oF runtime with multi-tenancy support.

Throughout our meticulous experiments, several significant observations emerged when comparing SPDK to NVMe-oPF: **1)** We observed a 2.94X increase in throughput and a 32.6% reduction in tail-latency for read workloads at 10 Gbps with 5 tenants per NVMe SSD. **2)** In the case of mixed read/write workloads at 100 Gbps, we achieved a 61.8% reduction in tail-latency with 5 tenants per SSD. **3)** Scaling up to 25 tenants on 5 SSDs, we achieved throughput improvements of 70% and 74.8% for write and 50:50 read/write workloads, respectively, on 100 Gbps. **4)** Across all workloads and experiments at 100 Gbps, we achieved a 39.3% increase in throughput while concurrently reducing latency by 17.3%. **5)** We observed linear scaling across all workloads at 100 Gbps. **6)** In application-level scaling experiments, NVMe-oPF demonstrated 25.2% increase in throughput for HDF5 [19] write workloads when scaling to 40 tenants on 4 NVMe SSDs.

In summary, this paper makes the following contributions:

❶ We address the challenges associated with coalescing methods for NVMe-oF request completions, successfully implementing efficient priority schemes along with multi-tenant support.

❷ We methodically design and assess NVMe-oPF request completion coalescing for the TCP/IP channel across 10, 25, and 100 Gbps fabrics, providing comprehensive insights into its performance.

❸ We introduce a user-friendly approach enabling applications in the userspace to utilize semantic data passing via flags, simplifying the implementation of advanced functionalities.

❹ We demonstrate the scalability achieved through multi-tenancy, showcasing the system's robust performance even with an increasing number of initiators.

❺ We exhibit application-level scalability, specifically with HDF5, emphasizing our proposed solution's practical applicability and versatility.

***To the best of our knowledge, NVMe-oPF is the first work to present a userspace NVMe-oF protocol that supports multiple tenants accessing single or many NVMe SSDs.***

## II. Background and Related Work

### A. Background

**NVMe-over-Fabrics** NVMe-over-Fabrics [14] (NVMe-oF) specification allows applications to use block storage protocol for accessing SSDs over networked fabrics such as Ethernet, Remote Direct Memory Access (RDMA), and Fibre Channel. NVMe-oF provides lower latency and higher throughput when compared to iSCSI [20], [21] for disaggregated storage within a data center. NVMe-oF defines the node that exposes NVMe-SSD over the network as the NVMe-oF target and the node that submits an I/O request to the NVMe-oF target as the NVMe-oF initiator. Since the underlying storage runtime is not multi-tenant aware, it treats all the I/O requests with the same priority.

**Intel SPDK** Intel's Storage Performance Development Kit [22] (SPDK) is a userspace library for developing high-performance storage applications. SPDK avoids costly system calls by moving the relevant drivers to userspace, uses polling to detect request completion, and eschews any locks in the I/O path. SPDK's NVMe driver allows the userspace application to issue concurrent I/O requests to the NVMe-SSD over the PCIe interface. It also consists of a userspace NVMe-oF target application that presents block devices over fabrics such as Ethernet, InfiniBand, and/or Fibre Channel and supports RDMA and TCP transports [23].

### B. Related Work

Both cloud [24]–[28] and HPC [29]–[33] communities have studied the benefits of sharing resources in the past. Burst Buffers (BBs) [29], [30], [34]–[36] are heavily being used in supercomputing environments to improve underlying storage system performance as perceived by end applications. Cao et al. [30] discuss the tradeoffs of employing local vs shared burst buffers and show the performance benefit shared burst buffer organizations bring to real HPC workloads.

A lot of work has been done to improve the submission rate of I/O requests to the NVMe device. Conway et al. [37] enhance key-value store performance on NVMe SSDs by providing more I/O concurrency using a new compaction policy and memtable design. Flashshare [38] aims to reduce interference from co-running workloads by propagating aspects of workload through the entire storage stack.

Other literature aims to improve remote storage performance where NVMe SSDs are connected to the client/application over the networked fabric. Klimovic et al. design ReFlex [39], a remote Flash storage system over Ethernet that combines networking and storage to provide low I/O latency and high throughput. Hwang et al. [40] draw inspiration from the networking domain to devise a kernel storage stack, blk-switch, for applications with varying needs. Tai et al. [41] showed the gains achieved by marking I/O requests as either latency-sensitive or throughput-critical and coalescing interrupts from local NVMe devices based on the request type. Gimbal [13] brings multi-tenancy to disaggregated NVMe devices using SmartNICs as a network switch to choose different NVMe devices depending on device condition. Peng et al. [42] introduces UMap to bring user-page management optimizations that are specific to application and storage characteristic needs using application hints. Gugnani et al. [43] introduce a QoS-aware NVMe-emulator that uses priority classes to provide SLA guarantees. An early case study observes performance improvements of multi-tenancy support which sheds light on the comprehensive design [12].

Our proposed designs differ from the existing studies in three ways. First, we propose priority schemes in the popular NVMe-oF specification to support various requirements from multi-tenants. Second, we analyze the benefits of the coalescing strategy in disaggregated storage settings with NVMe-oF for request completions rather than device interrupts. Third, we propose multiple enhancements in the popular SPDK library to achieve efficient priority schemes.

## III. Designing NVMe-oPF

This section presents our proposed designs for achieving efficient priority schemes in NVMe-oPF.

### A. Design Overview of NVMe-oPF

Supporting multi-tenancy for NVMe-oF requires a set of goals we aim to achieve – ***Goal 1***) Reduced congestion communication between NVMe-oF initiators and targets; ***Goal 2***) NVMe-oF target awareness for each initiator connected and respective optimization; and ***Goal 3***) Simple I/O optimization specification for user applications.

In this paper, we propose a new userspace NVMe-over-Priority-Fabrics (NVMe-oPF) runtime with respect to these goals. We fulfill those objectives with our multi-tenancy support which provides applications the ability to specify whether to optimize for latency or throughput. Our NVMe-oPF introduces three major concepts—request flags, priority managers, and throughput request queues. The request flags enable applications to tag I/O requests based on their performance goal. The NVMe-oF target creates specific queues for throughput-critical requests to avoid interference with different requests. The NVMe-oPF priority managers control request completion times and completion notification packets with respect to application optimization objectives. All three concepts make NVMe-oF multi-tenant aware.

The design is based on a coalescing strategy (explained in Section III-C), which is achieved by enhancing the Storage Performance Development Kit (SPDK) [44] on the NVMe-oF layer. Being in the userspace allows user applications easy access to specify between latency or throughput optimizations when sending I/O requests. We modify the NVMe-oF initiator and target TCP controllers to handle the additional priority flags and tenant information required for multi-tenant coalescing. The flags work alongside queues created for the NVMe-oPF initiator and target. The design of these queues allows the NVMe-oPF initiators and targets to solve issues present in standard NVMe-oF designs and more details are presented in Section IV.

Figure 2 shows the architecture of NVMe-oPF. We create efficient multi-tenancy by allowing multiple NVMe-oPF initiators to communicate to one target or to multiple targets while respecting individual

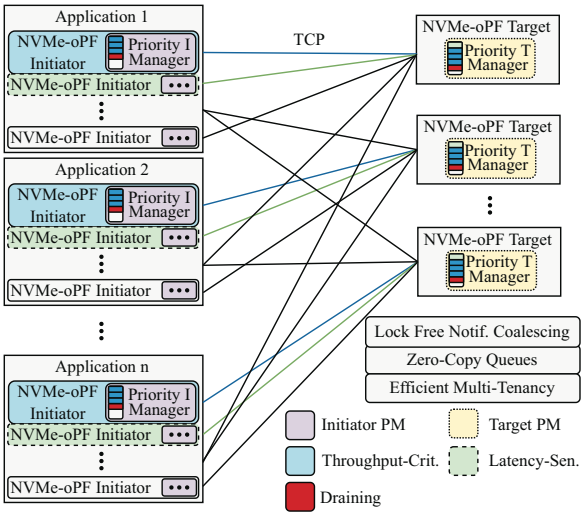



Fig. 2: Overview of NVMe-oPF architecture.

initiator optimizations. NVMe-oPF supports a range of different throughput-critical to latency-sensitive initiator ratios per target. In other words, regardless of the variety in traffic to one NVMe-oPF target, each initiator will be able to receive their respective desired performance optimization. The flow of these throughput-critical and latency-sensitive requests are handled by the Priority Manager (PM). In this way, we can effectively satisfy design Goal 1.

### *B. Priority-aware Data Flow*

For the NVMe-oF initiator and target to efficiently communicate to support multi-tenancy, both must contain logic that supports priority requests and coalescing. Figure 3 illustrates the data packet transfers between an initiator and target over four requests (both read and write have similar flow) for SPDK and NVMe-oPF. SPDK sends one completion notification per request regardless of the desired optimization. When an SPDK latency-sensitive initiator attempts to send a request concurrently with an SPDK throughput-critical initiator, the latency-sensitive request must wait for other requests to finish first.

Furthermore, the SPDK throughput-critical initiator will require processing time for completion notifications per request. The baseline lacks multi-tenancy support as each initiator will not receive their respective performance specifications. When an NVMe-oPF latency-sensitive initiator sends a request, it bypasses the queue on the target, and a completion notification is sent. The NVMe-oPF throughput-critical initiator marks the last request in a batch as drain and the target correspondingly executes all pending requests. Instead of sending four completion requests, only one will be sent to denote the completion of all preceding requests. Thus the NVMe-oPF throughput-critical initiator can reduce the time to process completion notifications.

In Figure 4(a), the diagram shows each request that is sent from the NVMe-oPF initiator application. Firstly the application sends each request to the NVMe-oPF target (along with a priority flag). The NVMe-oPF initiator will send a certain number (i.e., window

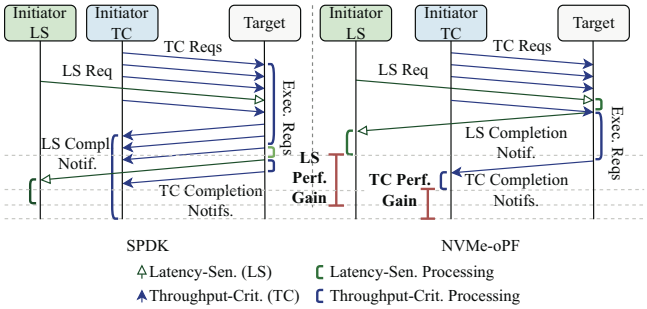
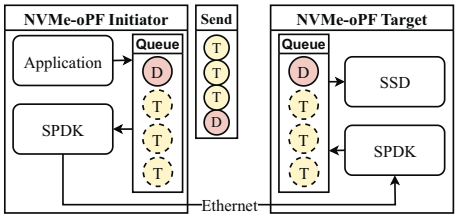
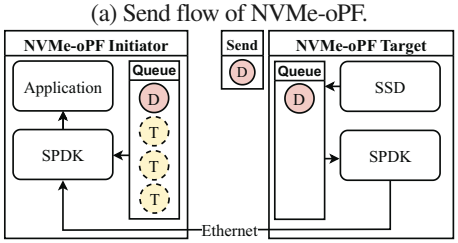



Fig. 3: Communication messages between NVMe-oF initiator and target for requests with a window size of four over TCP/IP. Arrows denote communication and brackets denote execution time respective to latency-sensitive (LS) or throughput-critical (TC) requests. The red brackets denote respective performance improvements for both LS and TC requests with NVMe-oPF.




(a) Send flow of NVMe-oPF.




(b) Completion flow of NVMe-oPF

Fig. 4: Send and completion flows of NVMe-oPF on a window size of 4 where throughput-critical (T) and drain (D) requests are sent.

size) of throughput-critical requests before sending one with a draining flag. For every sent request, both the NVMe-oPF initiator and target maintain a queue of all pending requests. Completion requests on the NVMe-oPF target will only be processed upon receiving a draining flag from the NVMe-oPF initiator.

In Figure 4(b), we can observe the return flow after the completion of all the requests in the queue. Sequentially, only one completion notification returns to the NVMe-oPF initiator per drain flag. In this way, we reduce the request completion network packet rate with respect to the set window size. In response to the NVMe-oPF target, the NVMe-oPF initiator acknowledges that the notification indicates the completion of all pending requests in the queue. Since the NVMe-oPF initiator has previously stored each request ID in its local queue, it can properly complete each preceding request without corresponding completion notifications.

Unlike throughput-critical, latency-sensitive requests do not have an additional queue. Latency-sensitive requests will process immediately upon arrival at the NVMe-oPF target regardless of the number of throughput-critical requests currently queued. Subsequently, a completion response from the NVMe-oPF target will be sent for the respective latency-sensitive request. In this way, our priority flow achieves a latency-optimized I/O specification.

Figure 5 (Alg:1) shows how the NVMe initiator creates and marks flags for throughput-critical and draining. We write the flags directly into the NVMe-oF Protocol Data Units (PDU). If the request is throughput-critical, we add it to the NVMe-oPF initiator queue.

When a response comes back from the NVMe-oPF target, Figure 5 (Alg:2) handles it. We loop through the queue of pending requests until the ID of the request matches with the received response. For each request in the queue, we mark them as complete. This way, all requests previously sent out from the NVMe-oPF initiator receive completion.

Figure 5 (Alg:3) shows when the NVMe-oPF target is ready to execute a request. Similar to Algorithm 1, we store throughput-critical requests in a queue, but on the NVMe-oPF target side. If the request is latency-sensitive, we will promptly transition the request to the execution state. If we are ready to drain the pending requests in the queue, we transition all requests in the queue into the execution state.

Once the requests have been sent to begin execution, Figure 5 (Alg:4) will begin. If the request sent is latency-sensitive, the request is completed immediately, a response is sent back to the NVMe-oPF initiator and we enter Algorithm 2. When the request is throughput-critical, completion logic is run. A response will only be sent to the NVMe-oPF initiator when the received request contains a draining flag.

Finally, our priority-aware data flow requires one more piece of information to allow NVMe-oPF priority managers to handle multi-tenancy. Each request will pass along a unique identifier per NVMe-oPF initiator connected to a target. This identifier informs NVMe-oPF target managers of the number of initiators and will properly handle priority data flow to service each initiator according to optimization desires. Thus, with all three types of semantic data, NVMe-oPF design can satisfy Goal 2.

### C. Coalescing Strategy

The coalescing strategy for increasing throughput involves completing the throughput-critical requests in batches rather than individually. The window size defines the number of requests that the initiator will send to the target before flushing out the queue of requests. In this way, we effectively reduce the amount of completions sent to the initiator.

To enable priority requests, we enrich each request with flags that denote the scheme with which to handle the request. These flags include the latency-sensitive flag, the throughput-critical flag, and the draining flag. These flags are embedded into the bits of each request and are read by the receivers.

**Latency-Sensitive:** To maintain low latency with our modifications, a request sent to the NVMe-oPF target can be flagged as latency-sensitive. This flag denotes that the request is to be completed and a response sent back immediately by bypassing throughput-critical queues.

**Throughput-Critical:** In optimizing throughput, it is necessary to coalesce multiple notification completions. The throughput-critical flag denotes that the request needs not be completed immediately to prioritize high-throughput. Requests with this flag are placed in a pending queue. Compared to latency-sensitive requests, the initiator may wait a certain period of time (dependent on I/O size and window size) before receiving a response back from the throughput-critical request at the tail of the queue.

**Draining:** The draining flag annotates a request to inform the target that all pending throughput-critical requests must be completed. Subsequently, after all requests are completed, the target replies to the initiator with a single completion notification response to denote the completion of all requests.

NVMe-oPF conveniently provides simple-to-use I/O flags for user applications. By easily passing a request with either latency-sensitive or throughput-critical flags, user applications can observe respective performance optimizations. For instance, if an application necessitates exchanging metadata or control information during a particular phase of the application, users can set requests as latency-sensitive. Conversely, during a high workload phase, users may prioritize throughput-critical requests. Our designs can be easily extended to support more I/O flags, which could represent other user application requirements. For the draining flag, the NVMe-oPF initiator sends it automatically according to the desired window size. With this, NVMe-oPF honors design Goal 3.

## IV. Implementation and Optimization

This section presents our software-level modifications and enhancements in Intel's open-source SPDK runtime library for NVMe-oPF.

The SPDK library is complex and can be difficult to locate points at which one should modify for coalescing. In our implementation, we effectively modify all areas of code to support NVMe initiator-target communication for NVMe-oPF. We modestly add three new flags (i.e., throughput-critical, latency-sensitive, and draining) into the spec files for the NVMe-oF initiator and target. Our queues and logic are built into SPDK and work hand-in-hand with SPDK's built-in functions and buffers.

### A. Lock-Free Optimization

The key optimizations for our lock-free implementation are independent queues for each throughput-critical NVMe-oPF initiator. For example, if there are two NVMe-oPF initiators, the NVMe-oPF target will contain two queues for throughput-critical requests. Thus, a single throughput-critical queue will not be shared between multiple NVMe-oPF initiators. Doing so would cause NVMe-oPF targets to respond to draining earlier than expected and flush back requests that may not be completed. Furthermore, if the window size of multiple initiators sums up to be larger than the queue depth of throughput-critical requests, request completions may never return and the NVMe-oPF initiator will lock. With isolated throughput-critical queues, we are lock-free and allow scaling. NVMe-oPF implements this by assigning multiple NVMe-oPF initiators with ID numbers. We write these ID numbers to the reserved bits in the request PDU so that on receipt, the NVMe-oPF target can differentiate requests. NVMe-over-TCP dictates network storage devices to

**Algorithm 1:** NVMe initiator algorithm: Before send

```
if request is throughput-critical
    PDU ← flag |= THRPUT_CRIT_MASK;
    queue[tail] ← req.cid;
    tail ← tail + 1;
if request is draining
    PDU ← flag |= DRAINING_MASK;
```

**Algorithm 2:** NVMe initiator algorithm: On response from NVMe target

```
if request is throughput-critical
    for int i = head; queue[i] != cid; i++ do
        req ← req.cid;
        Mark request as complete
        head ← head + 1;
    head ← head + 1;
```

**Algorithm 3:** NVMe target algorithm: Ready to execute request

```
if request is throughput-critical
    queue[head] ← req.cid;
    reqsQueued ← reqsQueued + 1;
else
    send to execution state
if (num requests queued > window size) || draining
    for all reqs queued do
        send to execution state
    reqsQueued ← 0;
```

**Algorithm 4:** NVMe target algorithm: Ready to complete request

```
if request is throughput-critical
    if request is in queue
        while req != null && req.state == transfer
            if request = draining
                send response to NVMe initiator
            else
                complete request but don't send
                 response
            incomplete ← incomplete + 1;
else
    complete request immediately
```

Fig. 5: Algorithms for NVMe-oPF multi-tenancy support.

communicate via PDU, thus there is no additional overhead as the reserved bits are transmitted regardless of their usage. We modestly use two reserved bits in the PDUs to pass latency-sensitive, throughput-critical, and draining flags. For initiator IDs, we use eight reserved bits in the PDU. As the size of the PDUs remains unchanged with our priority flags and initiator IDs, the parsing/processing time of PDUs on the target also remains almost the same.

### B. Zero-Copy Queues

Using a large number of queues that store requests can quickly burden both the NVMe-oPF initiator and target. Also, as multi-tenancy increases, the space complexity of the queues can become very large. Although we require queues for our design, we still maintain SPDK's zero-copy transport and low space complexity. This is because NVMe-oPF queues do not store requests but rather only each throughput-critical request's unique command identifier (CID) number. This allows NVMe-oPF queue space complexity to not increase drastically with increasing I/O request size, and/or number of tenants. It is to note, however, that the NVMe protocol will split single large I/O requests into multiple requests to be transferred in multiple packets over the network. Since NVMe-oPF reduces the number of completion notification packets, it will only further increase the point at which the interconnect becomes saturated compared to SPDK. Thus our implementation is zero-copy and scales well.

### C. Handling Out of Order Request Completions

Standard NVMe devices consist of two circular buffers to store requests that are sent to them [11]. Firstly, requests sent to the NVMe device are placed in one of the circular buffers called the Submission Queue (SQ). From the SQ, the NVMe controller places completed requests in the Completion Queue (CQ). However, the requests can be executed by the NVMe controller in any order which causes completions to be placed out of order. This can cause issues for reading completions from the CQ as it becomes difficult to track which request came first.

NVMe-oPF handles out-of-order completions with the same throughput-critical queues as previously mentioned. As shown in Figure 4, NVMe-oPF builds two throughput-critical queues on both the initiator and target. Thus for each NVMe initiator, each request sent to the NVMe-oPF target will be stored in its local queue. Since the NVMe-oPF initiator still contains the requests in its local queue (i.e., in their original order), it can mark their completions in the same order. This is because the NVMe-oPF initiator will match each completed request received from the target with those which is within the pending queue. Thus, completion times for each request will follow in the order they were queued. This allows us to handle out-of-order request completions while reducing completion notification packets.

### D. Optimized Window Size

Throughout our experiments, we found that window size cannot be static to achieve the highest throughput. We implement an optimized window size selection that will choose the correct window size based on certain parameters (i.e., workload type, initiator concurrency, TC/LS ratio). We optimize the window size to maximize throughput rather than reduce latency. This is because although window size does have a slight effect on latency, it is not significant in comparison to the improvement in throughput. If the user desires more adjustability, the window size can be dynamically changed during runtime after a draining request completion notification is received on the initiator. Thus, the user can quickly find the most optimal window size for their application. On the target, all pending requests before the draining request will be flushed, accommodating the initiator's dynamic window size.

## V. Evaluation

This section presents our evaluation results and analyzes the benefits of NVMe-oPF.

**Experimental Setup**: Our test environment uses two different hardware platforms. We use Chameleon Cloud's [45] storage_nvme nodes and Cloud Lab's [46] r6525 nodes (up to 10 nodes) to provide 10/25 Gbps and 100 Gbps Ethernet experiments, respectively. Note that according to the Top500 [47] list in Nov. 2023, 10/25/100 Gbps Ethernet networks are still among the most popular interconnect technologies (34.2%) being used in high-end machines and motivates our selection. Table I lists specifications for both Chameleon Cloud and Cloud Lab machines. All NVMe-oF initiators and targets use SPDK v20.07. We use SPDK's perf (SPDK's main benchmark), sending 4K sequential I/O requests for read, write, and mixed. Each experiment is run 5 times for 10 seconds and averaged. We also show application-level scaling performance with HDF5 with 4K read and write requests.

TABLE I: Experiment configuration. Hardware for Chameleon Cloud (CC) and CloudLab (CL) are shown.

| | CC | CL |
|---|---|---|
| Processor | AMD EPYC 7352 2.3GHz | AMD EPYC 7543 2.8GHz |
| Cores | 24 | 32 |
| RAM | 256GB | 256GB |
| NIC | 10/25 Gbps | 100 Gbps |
| SSD | 3.2 TB NVMe-SSD | 1.6 TB NVMe-SSD |

### A. Performance Analysis

For each test case, we observe multi-tenancy using different combinations of initiators. One or more initiators send requests with the latency-sensitive flag, and the other sends requests with the throughput-critical flag (we will refer to them as latency-sensitive and throughput-critical initiators). We observe the latency-sensitive initiator for latency performance, and for throughput performance, we observe the throughput-critical initiators. Our experiments show results when changing the ratio of latency-sensitive to throughput-critical initiators and how it may affect respective performance. Each throughput-critical and latency-sensitive initiator's queue depth is set to 128 and 1, respectively (including baseline SPDK).

We observed that three factors are needed to achieve performance improvement: 1) proper window size selection, 2) network speed, and 3) completion notification reduction, which is discussed in the following subsections.

*1) Window Size Selection:* Figure 6(a) shows the throughput and latency of NVMe-oPF over increasing window sizes. In this experiment, we observe performance results with multi-tenancy using one throughput-critical initiator and one latency-sensitive initiator, each running on individual nodes and communicating to an NVMe-oF target node. For NVMe-oPF, we can observe that as we increase the window size, we can more efficiently saturate the NVMe-oF target. NVMe-oPF achieves a peak throughput at a window size of 32 over 25/100 Gbps and performs 23.1% better than SPDK. Furthermore, NVMe-oPF maintains low latency with a slight increase in latency by 5.4% across all window sizes and network speeds. Generally, a large window size (e.g., 64) may not always be optimal, and a small window size (e.g., < 32) will not maximize throughput.

*2) Network Performance Impact:* Reducing the network bottleneck allows more requests and completion packets to be transmitted. Figure 6(b) shows the performance of one throughput-critical initiator communicating with one NVMe-oF target node with varying window sizes on 10/25/100 Gbps. Noticeably, both SPDK and NVMe-oPF are hindered in performance when using 10 Gbps speeds. NVMe-oPF's throughput gain does not increase with the window size. In this case, the network is the bottleneck as it is already saturated. For a window size of 64 at 10 Gbps, the completion notification packets begin to observe more delay before being received by the initiator, exacerbated by the network bottleneck and negatively impacting performance. If we observe with 25/100 Gbps, we can see an increase in throughput as window size increases. At a window size of 32 with 100 Gbps, NVMe-oPF increases throughput by 21.29%. Since this experiment only considers one initiator, we are limited by lack of multi-tenancy. By increasing concurrency on a large window size, we can further increase throughput gain. These experiments are shown in subsection V-D.

*3) Completion Notification Reduction:* Next, we analyze one of the root causes that impedes NVMe-oF throughput performance. We count the number of NVMe-oF request completions generated by the NVMe-oF target. The NVMe-oF target device is responsible for delivering a request completion notification per request. These large numbers of request completion notifications consume CPU processing at both the NVMe-oF target and initiator and generate a large number of network packets.

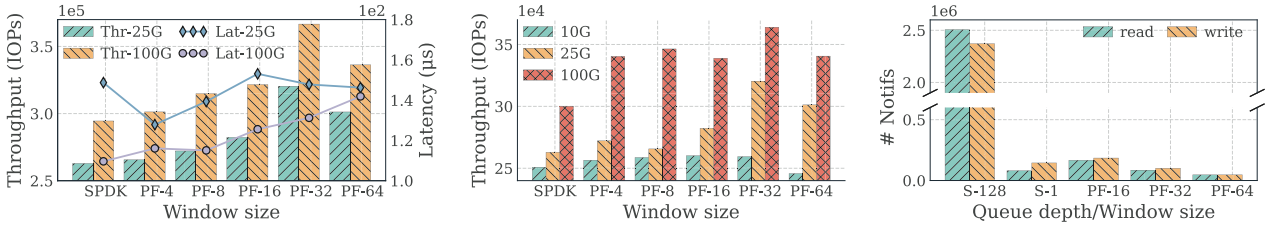
Figure 6(c) shows the number of read and write request completion notifications comparing baseline SPDK with NVMe-oPF. With a window size of 16 and a queue depth of 128, NVMe-oPF can reduce the number of outgoing request notifications significantly compared to SPDK at a queue depth of 128. With a window size of 32 and 64 at a queue depth of 128, NVMe-oPF reduces request completion notifications even past SPDK at a queue size of 1. Thus, NVMe-oPF can transmit fewer notification packets (i.e., while optimizing for throughput) than SPDK (i.e., while optimizing for latency). Previous experiments show that NVMe-oPF increases throughput while significantly reducing packet exchange over fabrics.

**Observation 1: With an optimized window size, NVMe-oPF with 100 Gbps increases throughput by 22.8% by reducing congestion of the interconnect and processing time of completions.**

### B. Throughput and Latency Concurrency

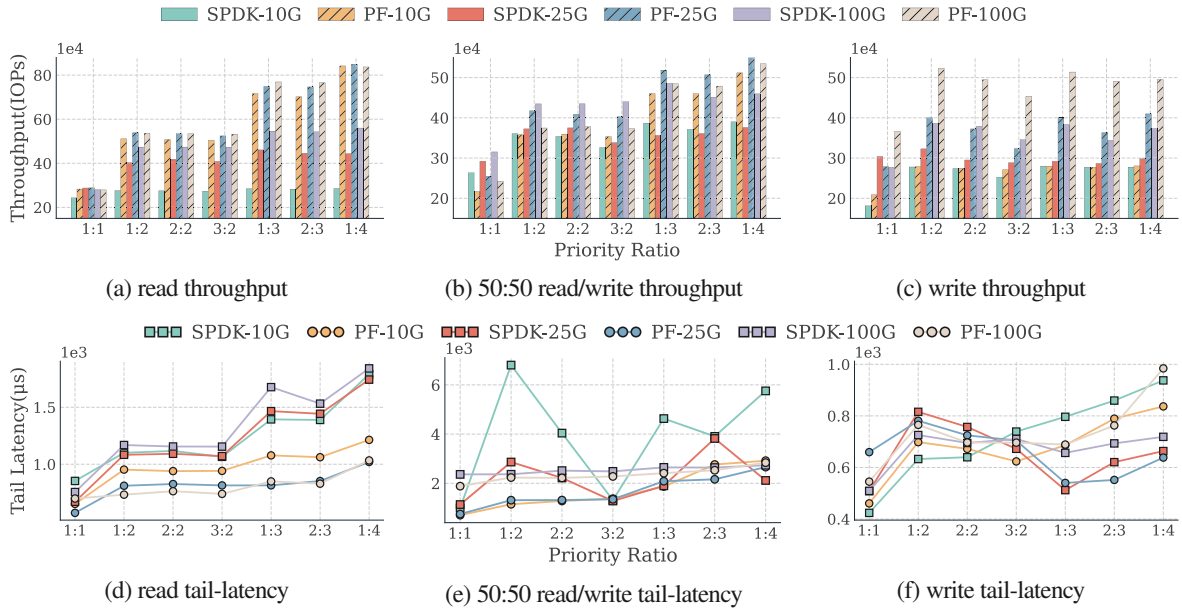
In a multi-tenancy scenario, there are at least two NVMe-oF initiators sending concurrent requests of distinct desired optimizations. Therefore, in the case of the baseline SPDK, requests are executed in a FIFO manner. With NVMe-oPF, requests are executed according to their denoted priority. To observe performance with multi-tenancy, we set up an experiment that scales to 5 NVMe-oF initiators with 7 different latency-throughput ratios (i.e., 1:1, 1:2, 2:2, 3:2, 1:3, 2:3, and 1:4) with the first number denoting the number of latency-sensitive initiators, and the second the number of throughput-critical initiators. These 7 ratios were selected to illustrate how NVMe-oPF handles introducing different priority ratios in terms of performance. The first ratio is a similar baseline of two total initiators as in previous experiments. Ratios 1:2, 2:2, and 3:2 maintain a constant number of throughput-critical initiators while increasing latency-sensitive initiators. In these ratios, we do not anticipate increasing aggregate throughput and we instead observe the effect on tail latency. Ratios 1:3, 2:3, and 1:4 are targeted at observing throughput scaling when adding more throughput-critical initiators.

Figure 7(a) shows the performance of sequential read requests with varying latency-throughput ratios. While increasing throughput, NVMe-oPF also provides lower tail-latency for latency-sensitive requests. For the priority ratio 1:1, we have one latency-sensitive and one throughput-critical initiator for two initiators sending requests to one target. We can observe slight benefits in throughput and latency for each respective NVMe-oPF initiator. As we increase the number of latency-sensitive initiators (1:2, 2:2, and 3:2), we can observe no effect on the throughput of NVMe-oF initiators, although we increased the number of tenants. We can observe an increase in throughput for NVMe-oPF initiators for every throughput-critical initiator added. NVMe-oPF at 10 Gbps shows a much larger increase




(a) Performance for 25/100 Gbps (b) Performance for 10/25/100 Gbps (c) Number of completion notifs.

Fig. 6: Analysis of the initial benefit of NVMe-oPF (PF) compared to SPDK (S). (a) shows throughput/latency performance across various window sizes with two initiators. (b) shows throughput performance impact of network speed across window sizes. (c) sums number of completion notifications generated.




(a) read throughput (b) 50:50 read/write throughput (c) write throughput

(d) read tail-latency (e) 50:50 read/write tail-latency (f) write tail-latency

Fig. 7: Performance on baseline SPDK and NVMe-oPF on 10/25/100 Gbps. Aggregate throughput and 99.99% tail latency are shown. The x-axis shows different latency-sensitive to throughput-critical initiators (LS:TC). Read, mixed 50:50 read/write, and write workloads are shown. Throughput performance is the aggregate of all throughput-critical initiators and tail-latency performance is respective to latency-sensitive initiators within each ratio.

in throughput than SPDK at 10 Gbps. SPDK at 10 Gbps shows a very similar throughput throughout all initiator ratios regardless of increasing throughput-critical initiators, whereas NVMe-oPF continues to increase in performance as we scale throughput-critical initiators. At a ratio of 1:4, NVMe-oPF has a peak performance improvement from SPDK of 194.5%. Although 10 Gbps is the slowest interconnect in our study, the combination of fast read requests and network congestion reduction allows NVMe-oPF to significantly optimize throughput. For NVMe-oPF with 25 Gbps, there is a significant throughput improvement in which NVMe-oPF achieves 91.3% higher throughput than SPDK at a ratio of 1:4. As with 10 and 25 Gbps, NVMe-oPF with 100 Gbps scales with high throughput at 49.5% increase from SPDK. Over 10/25/100 Gbps, we can observe that with 10 Gbps, NVMe-oPF has already saturated the target device with comparable performance to 25/100 Gbps. Thus, NVMe-oPF is a suitable solution to achieve performance similar to 100 Gbps with just 10 Gbps during highly concurrent multi-tenancy.

Figure 7(b) shows the performance of sequential 50:50 read/write requests with varying latency-throughput ratios. NVMe-oPF performs worse than in Figure 7(a). As we mix read/write workloads within window sizes, completion times of a batch of requests will begin to vary depending on the ratio of read/write requests. However, this is only an issue with fewer throughput-critical initiators. Since latency-sensitive requests can take longer than expected with write requests. We also still have comparable performance between 10/25/100 Gbps at a ratio of 1:4, which is similar to the peak achieved by NVMe-oPF write. This illustrates how NVMe-oPF reduces network congestion and allows high multi-tenancy scenarios to run with similar performance regardless of interconnect speed for read and mixed read/write workloads. Similar to the observations

in Figure 7(b), the NVMe device on the target is quickly saturated with NVMe-oPF maintaining slightly higher performance.

Figure 7(c) shows the performance of sequential write requests with varying latency-throughput ratios. For throughput, NVMe-oPF at 10 Gbps lacks performance benefit as the network becomes the bottleneck for write workloads compared to read. Read requests complete faster than write [28] and thus, NVMe-oPF with 10 Gbps lacks performance gain since processing write requests may bottleneck the interconnect. This is illustrated on 25 Gbps, as NVMe-oPF observes throughput performance improvement when increasing throughput-critical initiators to two. NVMe-oPF reduces completion notification processing time and can saturate at a higher throughput. Saturation is visible when observing ratios 1:2, 2:2, and 3:2 since, as we increase the number of latency-sensitive initiators, we can notice a gradual decrease in throughput performance. Although this decrease in performance is present; we can observe that NVMe-oPF with 25 Gbps maintains comparable performance (i.e., better with more tenants) to SPDK with 100 Gbps. Finally, NVMe-oPF with 100 Gbps at four throughput-critical initiators shows 32.6% throughput improvement from SPDK with 100 Gbps.

**Observation 2: For read and mixed read/write, NVMe-oPF reduces network congestion allowing 10/25/100 Gbps to all achieve similar high-throughput performance. Read workloads on 10 Gbps increases throughput by 2.94X compared to SPDK. For write workloads on 100 Gbps, NVMe-oPF improves throughput by 32.6% compared to SPDK.**

### C. Tail-Latency Studies

Due to the large queue sizes of the throughput critical initiators, latency-sensitive requests may be required to wait in multiple queues of over a size of 128 (i.e., queue depth) in default SPDK. This queue is further multiplied by the number of initiators in a multi-tenant environment. For this reason, increasing SPDK throughput-optimized initiators will only increase the queue size in front of a latency-sensitive request. Other factors, such as the conventional Ethernet interconnect, may cause variation in tail latency throughout the results. Figure 7(d) shows the tail latency of different latency-throughput ratios for read requests. Since read requests complete quickly, tail-latency is more consistent than in write workloads. However, any request by SPDK is subject to significant tail-latency increases due to back-of-the-line waiting. Each latency-sensitive initiator will suffer if any throughput-critical request incurs extra overhead to complete. Although this issue appears less often due to the completion rate of read requests, the improvement with NVMe-oPF in tail-latency is 38.8% at a ratio of 1:3 across all interconnects. Furthermore, as we increase multi-tenancy and interconnect speed, NVMe-oPF latency-sensitive initiators continue to achieve lower tail latency. Overall, NVMe-oPF tail latency is reduced compared to the baseline by an average of 29.1% across all seven latency-throughput ratios and interconnect speeds. More importantly, NVMe-oPF achieves a tail latency that does not increase significantly with varying latency-throughput ratio changes. This low tail latency is attributed to NVMe-oPF latency-sensitive requests that can quickly bypass the NVMe-oPF throughput-critical queue regardless of how large the queue becomes.

Figure 7(e) shows tail latency for a mixed 50:50 read/write workload. Now that write requests are introduced into the workload, further spikes in SPDK's tail-latency are visible. This trend appears specifically when the number of throughput-critical initiators is increased. There is observable variation in SPDK's tail-latency which is due to the 50:50 workload. Regardless, NVMe-oPF decreases tail-latency and its variation throughout all latency-throughput ratios. SPDK has large tail-latency peaks for each addition of throughput initiator which NVMe-oPF prevents. NVMe-oPF with 10 Gbps remains consistent throughout priority ratios and has a tail-latency performance improvement of 44.8% across all priority ratios. Over all ratios and interconnects, NVMe-oPF shows a 23.8% improvement in tail latency compared to SPDK.

Figure 7(f) shows tail latency for a write workload. At one throughput-critical initiator, SPDK with 10 Gbps performs the best with the lowest tail latency. This can be attributed to low interconnect usage and, consequently, low throughput, as shown in Figure 7(c). Once two throughput-critical initiators are added, the throughput for SPDK and NVMe-oPF no longer scales, and tail latency increases. This is because we have saturated the interconnect and can no longer benefit from scaling up. Compared to mixed read/write and read workloads, NVMe-oPF does not improve tail latency performance significantly. Since write requests require longer completion times, a single throughput-critical request can cause delays for an incoming latency-sensitive request. In this case, we can observe variation in the results. Ultimately, the scale presented in Figure 7(f) is an order of magnitude smaller than Figure 7(d,e), and thus this variation is minuscule. Across all ratios and interconnects, NVMe-oPF decreases tail-latency performance by only 2.6% from SPDK while achieving significant improvement in throughput. For all tail-latency studies, the 100 Gbps appears slower than slower interconnects. This can be attributed to the differences in NVMe devices across systems where the writes may perform slightly slower on the 100 Gbps as compared to the 10 and 25 Gbps.

**Observation 3: NVMe-oPF attains low tail-latency with numerous tenants and enhances performance in terms of tail latency by 25.6% across all tenant-priority ratios and interconnect speeds. NVMe-oPF achieves a peak tail-latency reduction of 81.1%.**

### D. Scale-Out Studies

We observe the performance benefits of scaling the number of nodes. We include two experimental setups - 1) a total of 10 nodes (5 initiator-nodes and 5 target-nodes) with an increasing number of initiators per initiator-node (scaling pattern 1), and 2) the same 10 nodes as setup 1, but we instead fix the number of initiators per initiator-node and increase the number of initiator-nodes (scaling pattern 2). For setup 1, each initiator-node is connected to a single target-node. Between the two nodes, the initiator-node communicates up to 5 initiators to 1 target on the target-node. Setup 2 communicates 4 initiators to 1 target (LS:TC ratio is 0:4) between the initiator-node and target-node. We observe the performance benefits during multi-tenancy and scale-out with these two experiment setups. In setup 1, we observe the number of initiators at which a single target node becomes saturated. In setup

2, we use the optimal number of initiators per target and scale up nodes. For both setups, each node runs at 100 Gbps speeds.

We show scaling experiments with three workloads: read, mixed read/write, and write. Figure 8(a) shows a read workload for scaling the number of initiators per node with 100 Gbps. We can observe that SPDK quickly plateaus in throughput performance at 15 initiators and begins slowly degrading in performance. For NVMe-oPF, the throughput performance continues to increase past 15 total initiators. Informing us that SPDK can not saturate the target when increasing initiators to 25. Across all numbers of initiators, NVMe-oPF gives performance improvements of 27.2% and 12.8% for throughput and latency, respectively.

Figure 8(b) shows a mixed 50:50 read/write workload for scaling the number of initiators per node with 100 Gbps. At 10 initiators, NVMe-oPF does not perform better than SPDK. As observed in Figure 7(b), mixed workloads will require more tenants to achieve performance benefits fully. Thus, as we scale the number of initiators, the performance gap between NVMe-oPF and SPDK becomes greater with 74.8% and 68.8% improvement in throughput and latency performance for NVMe-oPF. Figure 8(c) shows a write workload for scaling the number of initiators per node with 100 Gbps. At 15 initiators, both SPDK and NVMe-oPF can no longer improve scaling performance. As average latency continuously increases with scaling, the interconnect has likely saturated. However, NVMe-oPF increases the throughput at the point of saturation and peaks at 25 initiators. NVMe-oPF increases throughput performance by 64.3% when over 10 initiators while maintaining low latency with a gain of only 2.2% overall.

For scaling pattern 2, we use the same experiment workload types. Figure 8(d) shows a read workload for scaling the number of nodes with 100 Gbps. Both SPDK and NVMe-oPF scale well when increasing the number of nodes. NVMe-oPF has a slight improvement in throughput 19.6% across all numbers of initiators compared to SPDK. Figure 8(e) shows a mixed 50:50 workload for scaling the number of nodes with 100 Gbps. SPDK scaling performance suffers as completion rate has been slowed down due to the addition of write requests. NVMe-oPF scales well compared to SPDK performance, as NVMe-oPF performance continues to increase. Across all number of initiators, NVMe-oPF improves performance by 61.3% compared to SPDK. Figure 8(f) shows a write workload for scaling the number of nodes with 100 Gbps. Observing SPDK throughput performance at 16 initiators and up illustrates over saturation of the interconnect and decreases throughput per initiator. Since NVMe-oPF reduces network utilization, NVMe-oPF can scale well past the point where SPDK plateaued in throughput performance. Across all number of initiators, NVMe-oPF throughput increased by 95.2%.

**Observation 4: Due to the performance difference between read and write, SPDK cannot scale well past 16 initiators, whereas NVMe-oPF continues to scale throughput performance. Across all number of initiators, NVMe-oPF throughput is increased by 95.2% for write workloads.**

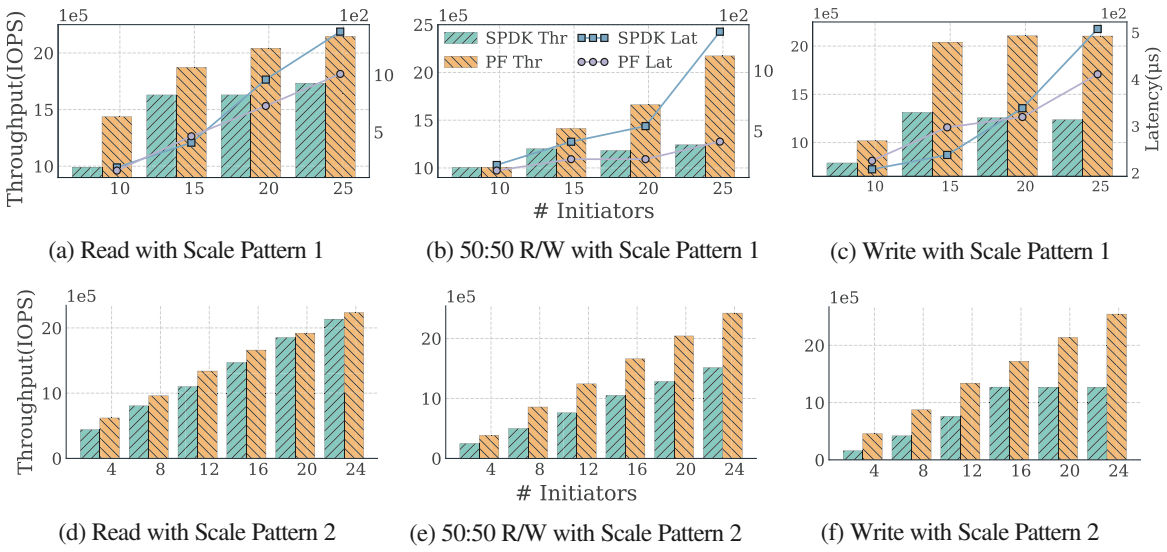
### *E. Application-Level Scalability Evaluation with HDF5*

*1) h5bench:* HDF5 [19] is a hierarchical data format specification with library implementation for fast parallel I/O and threading. HDF5 is widely used for scientific applications within HPC environments. It is the most used library for performing parallel I/O at the National Energy Research Scientific Computing Center (NERSC) and is among the top at several US Department of Energy (DOE) supercomputing facilities [48]. Considering HDF5's parallel I/O and multi-process support, we observe how NVMe-oPF's multi-tenancy scheme can further benefit HDF5. We co-design a popular benchmark representing HDF5 I/O kernels (h5bench) with NVMe-oPF. This is achieved with the HDF5 Virtual Object Layer (VOL) to intercept HDF5 APIs and utilize NVMe-oPF priority managers. Our results in HDF5 enable users to use NVMe-oPF in any real-world dataset they would run on HDF5.

We run h5bench's read and write I/O kernels from multiple clients. Our benchmark configuration writes 8*1024*1024 (8M) particles for one 1D array stored as one HDF5 dataset. Our h5bench reads are configured similarly. We mimic SPDK's perf benchmark, with each I/O access at a read/write size of 4K. For scaling tests, we replicate our previous scaling experiments (i.e., scaling patterns 1 and 2) as shown in Figure 8 and we use 8 total nodes (4 initiator-nodes and 4 target-nodes). We use Open MPI to parallelize multiple ranks with each rank in HDF5 hosting an initiator (we refer to initiators interchangeably with ranks). For scaling pattern 1, we increase the number of initiators per initiator-node up to 10 initiators. For scaling pattern 2, we set the number of initiators at 10 while increasing the number of initiator-nodes. In both scenarios, we effectively show scaling performance for up to 40 initiators on 4 targets. Each initiator-node will have one latency-sensitive initiator and the rest as throughput-critical initiators to evaluate both latency and throughput.

Figure 9(a) depicts how NVMe-oPF bandwidth improvement from SPDK continues to increase as more initiator-nodes are increased for a write workload. We can observe that NVMe-oPF improves performance by 25.2% while maintaining low latency with SPDK at 40 tenants. This illustrates that reducing completion notifications will reduce the number of network packets transmitted over fabrics for HDF5 applications. Figure 9(b) shows read performance on the same experiment. The performance improvement brought by NVMe-oPF over SPDK for read workload is low compared to write workload. We attribute this result to an internal data loading pattern within h5bench. We still observe the average latency of read requests is lower than write, which further illustrates the overhead resulting from dataset loading in h5bench. However, NVMe-oPF performance improvement from SPDK is similar to writing in which improvement increases with increasing initiator-nodes. Figure 9(c) illustrates the performance of an increasing number of initiators per initiator-node on a write workload. We can observe that 7 initiators per initiator-node (28 total initiators) can saturate the target-nodes. Although performance begins to drop; we can still obtain performance improvement from NVMe-oPF.

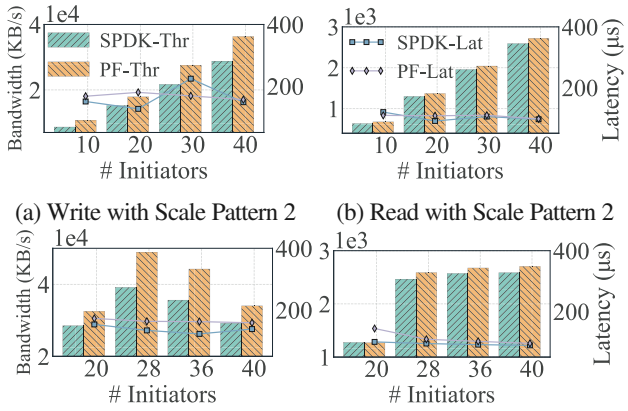
Figure 9(d) shows the same saturation point at 28 total initiators. Since read requests are completed quicker than write requests, we maintain performance after over-saturating our target. After saturation, NVMe-oPF shows better performance than SPDK.




(a) Read with Scale Pattern 1 (b) 50:50 R/W with Scale Pattern 1 (c) Write with Scale Pattern 1

(d) Read with Scale Pattern 2 (e) 50:50 R/W with Scale Pattern 2 (f) Write with Scale Pattern 2

Fig. 8: Scale-out performance on baseline SPDK and NVMe-oPF on 100 Gbps. A total of 10 nodes are run where 5 act as initiator nodes and the rest as target nodes. (a, b, c) shows results for 5 initiators per initiator-node and (d, e, f) shows results for 4 initiators per node.




(a) Write with Scale Pattern 2 (b) Read with Scale Pattern 2

(c) Write with Scale Pattern 1 (d) Read with Scale Pattern 1

Fig. 9: Performance of h5bench I/O kernels when scaling-out NVMe-oF initiators on SPDK and NVMe-oPF with 25 Gbps on read and write workloads.

**Observation 5: HDF5 with NVMe-oPF increases write throughput by up to 25.2% when scaling to 40 tenants over 4 NVMe SSDs on 100 Gbps**

**Discussion on h5bench overhead** As explained in prior experiment results, h5bench's dataset loading for read requests increases overall runtime. Unlike write, h5bench read must perform dataset loading overheads between read requests (h5bench timesteps). For this reason, the overall bandwidth is much lower than write in our results. While our results still represent the performance of SPDK and NVMe-oPF on HDF5, we will work further with the HDF5 community to improve these results.

## VI. Conclusion and Future Work

NVMe-oPF provides the ability for workloads to specify their I/O optimization strategies—latency-sensitive or throughput-critical—and makes the NVMe-oF runtime multi-tenant aware. NVMe-oPF creates a priority scheme for numerous tenants with differentiable optimization strategies by honoring three goals–1) efficient communication by all tenants and targets; 2) tenant priority optimization awareness across all tenants; and 3) easy specification of I/O optimization for applications. We augment userspace NVMe-oF runtime with zero-copy queues and priority managers to be aware of application needs. We also introduce three new request flags in the NVMe-oF specification and enable support for multi-tenancy in the NVMe-oF target. Our experiments indicate that NVMe-oPF with 10 Gbps increases the throughput of remote read I/O requests by 2.94X and reduces tail-latency by 32.6% with 5 tenants. For remote write I/O requests, NVMe-oPF with 100 Gbps improves throughput by 32.6% while maintaining low latency compared to SPDK. When scaling to 25 tenants, NVMe-oPF with 100 Gbps increases throughput by 70% and 74.8% on write and mixed read/write workloads respectively. For application-level experiments, NVMe-oPF increases the throughput of HDF5 by 25.2% for write workloads when scaling to 40 tenants over 4 NVMe SSDs on 100 Gbps.

In the future, we will co-design more HPC and datacenter applications with the proposed NVMe-oPF schemes to demonstrate the benefits. We hope to further collaborate with the HDF5 community to bring further benefit with NVMe-oPF.

## Acknowledgments

We are grateful to our anonymous reviewers for their invaluable feedback on the paper. We would like to thank our lab mates, with special appreciation to Weicong Chen and Yuke Li for their valuable input. We gratefully acknowledge the computing resources provided on Chameleon Cloud and CloudLab which provide us with our testbed for all our experiments. This work was supported in part by NSF research grants OAC #2321123 and #2340982 and a DOE research grant DE-SC0024207.

## References


[1] X. Lu and A. Kashyap, "Towards Offloadable and Migratable Microservices on Disaggregated Architectures: Vision, Challenges, and Research Roadmap," in *The Second Workshop On Resource Disaggregation and Serverless (WORDS'21), co-located with ASPLOS 2021*, WORDS '21, ACM, 2021. Vision Paper.

[2] P. X. Gao, A. Narayan, S. Karandikar, J. Carreira, S. Han, R. Agarwal, S. Ratnasamy, and S. Shenker, "Network Requirements for Resource Disaggregation," in *12th USENIX symposium on operating systems design and implementation (OSDI 16)*, pp. 249–264, 2016.

[3] Y. Shan, Y. Huang, Y. Chen, and Y. Zhang, "{LegoOS}: A Disseminated, Distributed {OS} for Hardware Resource Disaggregation," in *13th USENIX Symposium on Operating Systems Design and Implementation (OSDI 18)*, pp. 69–87, 2018.

[4] S. Han, N. Egi, A. Panda, S. Ratnasamy, G. Shi, and S. Shenker, "Network Support for Resource Disaggregation in Next-Generation Datacenters," in *Proceedings of the Twelfth ACM Workshop on Hot Topics in Networks*, pp. 1–7, 2013.

[5] Q. Zhang, Y. Cai, X. Chen, S. Angel, A. Chen, V. Liu, and B. T. Loo, "Understanding The Effect of Data Center Resource Disaggregation on Production DBMSs," *Proceedings of the VLDB Endowment*, vol. 13, no. 9, 2020.

[6] G. Michelogiannakis, B. Klenk, B. Cook, M. Y. Teh, M. Glick, L. Dennison, K. Bergman, and J. Shalf, "A Case for Intra-Rack Resource Disaggregation in HPC," *ACM Transactions on Architecture and Code Optimization (TACO)*, vol. 19, no. 2, pp. 1–26, 2022.

[7] A. Kashyap, S. Gugnani, and X. Lu, "Impact of Commodity Networks on Storage Disaggregation with NVMe-oF," in *Proceedings of International Symposium on Benchmarking, Measuring, and Optimizing* (W. Gao, J. Zhan, and F. Wolf, eds.), Bench '20, (Cham), Springer International Publishing, 2020.

[8] C. Guo, X. Wang, G. Shen, S. Bose, J. Xu, and M. Zukerman, "Exploring the Benefits of Resource Disaggregation for Service Reliability in Data Centers," *IEEE Transactions on Cloud Computing*, 2022.

[9] S. Angel, M. Nanavati, and S. Sen, "Disaggregation and the Application," in *12th USENIX Workshop on Hot Topics in Cloud Computing (HotCloud 20)*, 2020.

[10] D. Gouk, S. Lee, M. Kwon, and M. Jung, "Direct Access,{High-Performance} Memory Disaggregation with {DirectCXL}," in *2022 USENIX Annual Technical Conference (USENIX ATC 22)*, pp. 287–294, 2022.

[11] "NVM ExpressTM over Fabrics Revision 1.1," 2019.

[12] D. Ng, C. Parkinson, A. Lin, A. Kashyap, and X. Lu, "An Early Case Study with Multi-Tenancy Support in SPDK's NVMe-over-Fabric Designs," in *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*, SC '23, (New York, NY, USA), Association for Computing Machinery, Nov. 2023. Research Poster Paper.

[13] J. Min, M. Liu, T. Chugh, C. Zhao, A. Wei, I. H. Doh, and A. Krishnamurthy, "Gimbal: Enabling Multi-Tenant Storage Disaggregation on SmartNIC JBOFs," in *Proceedings of the 2021 ACM SIGCOMM 2021 Conference*, pp. 106–122, 2021.

[14] "NVMe-over-Fabrics Specification," 2023.

[15] J. Liu, Q. Koziol, G. F. Butler, N. Fortner, M. Chaarawi, H. Tang, S. Byna, G. K. Lockwood, R. Cheema, K. A. Kallback-Rose, D. Hazen, and M. Prabhat, "Evaluation of HPC Application I/O on Object Storage Systems," in *2018 IEEE/ACM 3rd International Workshop on Parallel Data Storage Data Intensive Scalable Computing Systems (PDSW-DISCS)*, pp. 24–34, 2018.

[16] J. Soumagne, J. Henderson, M. Chaarawi, N. Fortner, S. Breitenfeld, S. Lu, D. Robinson, E. Pourmal, and J. Lombardi, "Accelerating HDF5 I/O for Exascale Using DAOS," *IEEE Transactions on Parallel and Distributed Systems*, vol. 33, no. 4, pp. 903–914, 2022.

[17] M. S. Breitenfeld, N. Fortner, J. Henderson, J. Soumagne, M. Chaarawi, J. Lombardi, and Q. Koziol, "DAOS for Extreme-scale Systems in Scientific Applications," 2017.

[18] Intel, "DAOS: Revolutionizing High-Performance Storage." https://www.intel.com/content/www/us/en/high-performance-computing/daos-high-performance-storage-brief.html, 2023.

[19] The HDF Group, "Hierarchical Data Format, version 5," 1997-NNNN. https://www.hdfgroup.org/HDF5/.

[20] H. M. Khosravi, Abhijeet Joglekar, and Ravi Iyer, "Performance Characterization of iSCSI Processing in a Server Platform," in *PCCC 2005. 24th IEEE International Performance, Computing, and Communications Conference, 2005.*, pp. 99–107, 2005.

[21] A. Joglekar, M. E. Kounavis, and F. L. Berry, "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, (USA), p. 20, USENIX Association, 2005.

[22] Z. Yang, J. R. Harris, B. Walker, D. Verkamp, C. Liu, C. Chang, G. Cao, J. Stern, V. Verma, and L. E. Paul, "SPDK: A Development Kit to Build High Performance Storage Applications," in *2017 IEEE International Conference on Cloud Computing Technology and Science (CloudCom)*, pp. 154–161, 2017.

[23] "SPDK: NVMe over Fabrics Target," 2023.

[24] H. AlJahdali, A. Albatli, P. Garraghan, P. Townend, L. Lau, and J. Xu, "Multi-tenancy in Cloud Computing," in *2014 IEEE 8th International Symposium on Service Oriented System Engineering*, pp. 344–351, 2014.

[25] M. Factor, D. Hadas, A. Harnama, N. Har'el, E. K. Kolodner, A. Kurmus, A. Shulman-Peleg, and A. Sorniotti, "Secure Logical Isolation for Multi-tenancy in Cloud Storage," in *2013 IEEE 29th Symposium on Mass Storage Systems and Technologies (MSST)*, pp. 1–5, 2013.

[26] D. Banks, J. Erickson, M. Rhodes, and J. Erickson, "Multi-Tenancy in Cloud-Based Collaboration Services," *Information Systems Journal*, 2009.

[27] J. M. A. Calero, N. Edwards, J. Kirschnick, L. Wilcock, and M. Wray, "Toward a Multi-Tenancy Authorization System for Cloud Services," *IEEE Security Privacy*, vol. 8, no. 6, pp. 48–55, 2010.

[28] A. Kashyap and X. Lu, "NVMe-oAF: Towards Adaptive NVMe-oF for IO-Intensive Workloads on HPC Cloud," in *Proceedings of the 31st International Symposium on High-Performance Parallel and Distributed Computing (HPDC)*, pp. 56–70, 2022.

[29] H. Khetawat, C. Zimmer, F. Mueller, S. Atchley, S. S. Vazhkudai, and M. Mubarak, "Evaluating Burst Buffer Placement in HPC Systems," in *2019 IEEE International Conference on Cluster Computing (CLUSTER)*, pp. 1–11, 2019.

[30] L. Cao, B. W. Settlemyer, and J. Bent, "To Share or Not to Share: Comparing Burst Buffer Architectures," in *Proceedings of the 25th High Performance Computing Symposium*, HPC '17, (San Diego, CA, USA), Society for Computer Simulation International, 2017.

[31] T. Al-Jody, H. Aagela, and V. Holmes, "Inspiring the Next Generation of HPC Engineers with Reconfigurable, Multi-Tenant Resources for Teaching and Research," *Sustainability*, vol. 13, no. 21, 2021.

[32] F. Zahid, E. G. Gran, B. Bogdański, B. D. Johnsen, and T. Skeie, "Efficient Network Isolation and Load Balancing in Multi-Tenant HPC Clusters," *Future Generation Computer Systems*, vol. 72, pp. 145–162, 2017.

[33] S. Gugnani, T. Li, and X. Lu, "NVMe-CR: A Scalable Ephemeral Storage Runtime for Checkpoint/Restart with NVMe-over-Fabrics," in *2021 IEEE International Parallel and Distributed Processing Symposium (IPDPS)*, pp. 172–181, IEEE, 2021.

[34] D. Kimpe, K. Mohror, A. Moody, B. Van Essen, M. Gokhale, R. Ross, and B. R. de Supinski, "Integrated In-System Storage Architecture for High Performance Computing," in *Proceedings of the 2nd International Workshop on Runtime and Operating Systems for Supercomputers*, ROSS '12, (New York, NY, USA), Association for Computing Machinery, 2012.

[35] T. Wang, S. Oral, M. Pritchard, B. Wang, and W. Yu, "TRIO: Burst Buffer Based I/O Orchestration," in *2015 IEEE International Conference on Cluster Computing*, pp. 194–203, 2015.

[36] D. Shankar, X. Lu, and D. K. Panda, "Boldio: A Hybrid and Resilient Burst-Buffer over Lustre for Accelerating Big Data I/O.," in *Proceedings of IEEE International Conference on Big Data* (J. Joshi, G. Karypis, L. Liu, X. Hu, R. Ak, Y. Xia, W. Xu, A.-H. Sato, S. Rachuri, L. H. Ungar, P. S. Yu, R. Govindaraju, and T. Suzumura, eds.), BigData '16, pp. 404–409, IEEE Computer Society, 2016. Short Paper.

[37] A. Conway, A. Gupta, V. Chidambaran, M. Farach-Colton, R. Spillane, A. Tai, and R. Johnson, "SplinterDB: Closing the Bandwidth Gap for NVMe Key-Value Stores," in *Proceedings of the 2020 USENIX Conference on Usenix Annual Technical Conference*, USENIX ATC'20, (USA), USENIX Association, 2020.

[38] J. Zhang, M. Kwon, D. Gouk, S. Koh, C. Lee, M. Alian, M. Chun, M. T. Kandemir, N. S. Kim, J. Kim, and M. Jung, "Flashshare: Punching through Server Storage Stack from Kernel to Firmware for Ultra-Low Latency SSDs," in *Proceedings of the 13th USENIX Conference on Operating Systems Design and Implementation*, OSDI'18, (USA), p. 477–492, USENIX Association, 2018.

[39] A. Klimovic, H. Litz, and C. Kozyrakis, "ReFlex: Remote Flash Local Flash," in *Proceedings of the Twenty-Second International Conference on Architectural Support for Programming Languages and Operating Systems*, ASPLOS '17, (New York, NY, USA), p. 345–359, Association for Computing Machinery, 2017.

[40] J. Hwang, M. Vuppalapati, S. Peter, and R. Agarwal, "Rearchitecting Linux Storage Stack for $\mu$s Latency and High Throughput," in *15th {USENIX} Symposium on Operating Systems Design and Implementation ({OSDI} 21)*, pp. 113–128, 2021.

[41] A. Tai, I. Smolyar, M. Wei, and D. Tsafrir, "Optimizing Storage Performance with Calibrated Interrupts," *ACM Transactions on Storage (TOS)*, vol. 18, no. 1, pp. 1–32, 2022.
[42] I. Peng, M. McFadden, E. Green, K. Iwabuchi, K. Wu, D. Li, R. Pearce, and M. Gokhale, "UMap: Enabling Application-Driven Optimizations for Page Management," in *2019 IEEE/ACM Workshop on Memory Centric High Performance Computing (MCHPC)*, pp. 71–78, IEEE, 2019.
[43] S. Gugnani, X. Lu, and D. K. Panda, "Analyzing, Modeling, and Provisioning QoS for NVMe SSDs," in *2018 IEEE/ACM 11th International Conference on Utility and Cloud Computing (UCC)*, pp. 247–256, IEEE, 2018.
[44] Z. Yang, J. R. Harris, B. Walker, D. Verkamp, C. Liu, C. Chang, G. Cao, J. Stern, V. Verma, and L. E. Paul, "SPDK: A Development Kit to Build High Performance Storage Applications," in *2017 IEEE International Conference on Cloud Computing Technology and Science (CloudCom)*, pp. 154–161, IEEE, 2017.
[45] Chameleon Cloud, "Chameleon Cloud." https://www.chameleoncloud.org/, 2023.
[46] D. Duplyakin, R. Ricci, A. Maricq, G. Wong, J. Duerig, E. Eide, L. Stoller, M. Hibler, D. Johnson, K. Webb, A. Akella, K. Wang, G. Ricart, L. Landweber, C. Elliott, M. Zink, E. Cecchet, S. Kar, and P. Mishra, "The Design and Operation of CloudLab," in *Proceedings of the USENIX Annual Technical Conference (ATC)*, pp. 1–14, July 2019.
[47] Top500, "Top500." https://www.top500.org/statistics/list/, 2023.
[48] S. Byna, M. S. Breitenfeld, B. Dong, Q. Koziol, E. Pourmal, D. Robinson, J. Soumagne, H. Tang, V. Vishwanath, and R. Warren, "ExaHDF5: Delivering Efficient Parallel I/O on Exascale Computing Systems," *Journal of Computer Science and Technology*, vol. 35, pp. 145–160, 2020.