LPNS: Scalable and Latency-Predictable Local Storage Virtualization for Unpredictable NVMe SSDs in Clouds

Bo Peng, Cheng Guo, Jianguo Yao, and Haibing Guan,
Shanghai Jiao Tong University

https://www.usenix.org/conference/atc23/presentation/peng

This paper is included in the Proceedings of the
2023 USENIX Annual Technical Conference.

July 10–12, 2023 • Boston, MA, USA

978-1-939133-35-9

LPNS: Scalable and Latency-Predictable Local Storage Virtualization for Unpredictable NVMe SSDs in Clouds

Bo Peng
Shanghai Jiao Tong University

Cheng Guo
Shanghai Jiao Tong University

Jianguo Yao
Shanghai Jiao Tong University

Haibing Guan
Shanghai Jiao Tong University

Abstract

Latency predictability of storage is one important QoS target of the public clouds. Although modern storage virtualization techniques are devoted to providing fast and scalable storage for clouds, these works usually concentrate exclusively on giving high IOPS throughput without eliminating the device-level interference between multi-tenant virtualized devices and providing latency predictability for cloud tenants when the cloud infrastructures virtualize millions of the current commercially-available but unpredictable NVMe SSDs.

To resolve this issue, we propose a novel local storage virtualization system called LPNS to provide latency-predictable QoS control for hybrid-deployed local cloud storage, including virtualized machines, containers, and bare-metal cloud services. The OS-level NVMe virtualization LPNS designs reliable self-feedback control, flexible I/O queue and command scheduling, scalable polling design, and involves a deterministic network calculus-based formalization method to give upper bounds to virtualized device latency. The evaluation demonstrates that LPNS can achieve up to 18.72× latency optimization of the mainstream NVMe virtualization with strong latency bounds. LPNS can also increase up to 1.45× additional throughput and a better latency bound than the state-of-the-art storage latency control systems.

1 Introduction

Storage virtualization [23, 63] is critical to optimize limited hardware utilization and simplify storage management by providing consistent and straightforward I/O management interfaces in cloud systems. Since NVMe devices deployed in cloud platforms are usually inadequately utilized in terms of throughput [34, 35, 52], most previous works concentrated exclusively on achieving high-throughput targets, including software-level virtualization such as SPDK [25] and MDev-NVMe [55], the hardware-assisted virtualization such as the direct pass-through [70] and Single Root I/O Virtualization (SR-IOV) [14], and hardware/software co-design researches such as LeapIO [41] and FVM [37]. However, when more latency-critical businesses, in addition to the throughput-intensive businesses, have been migrating to the public cloud [12] for performance benefit, a fundamental contradiction between predictable latency and efficiency of the device sharing occurs when cloud platforms integrate storage virtualization: On the one hand, cloud service providers (CSP) tend to oversubscribe infrastructures by sharing them among multiple tenants to achieve better Input/Output Operations Per Second (IOPS) performance and higher energy efficiency [28, 44, 45]. On the other hand, the latency-critical tenants expect exclusive performance to ensure the latency bound of their services without worrying about interference from other guest machines that share the same NVMe SSD, i.e., latency-predictable Quality of Service (QoS).

The mainstream storage solutions can usually ensure high total throughput [25, 37, 41, 55, 57, 70] or fair bandwidth sharing [21, 71] but lack support for latency performance isolation between multi-tenant virtualized storage. For example, we quote a contention scenario where two virtual machines (VM1, VM2) share one Optane P5800X SSD [24] by using SPDK, and we use Figure 1 to show how the interference between these VMs hurts the average latency performance. In VM1, an IOPS-lightweight but latency-sensitive workload runs with a service level of agreement (SLA) at 30μs latency, while VM2 generates a throughput-intensive workload every 10 seconds as a competitor. Unfortunately, VM1 suffers a severe performance thrashing of over 250% additional latency as the workload weight of VM2 fluctuates, missing the SLA during its running time. The reason for this phenomenon is that the general storage controller naively handles all hardware queues used by different processes in a round-robin fashion [66], so the hardware queues are saturated with the I/O commands of the workload $w2$, resulting in a long queue operation time and unpredictable latency of $w1$.

In order to solve the performance interference issue and provide latency-predictable QoS, we propose LPNS, an NVMe virtualization solution that provides latency-predictable virtualized storage in NVMe virtualization and sharing scenarios. LPNS replaces the original static I/O queue allocation be-

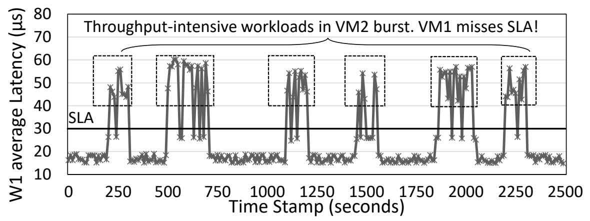


Figure 1: The workload $w1$ in the VM1 interfered by intensive workloads $w2$ in the VM2 sharing the same SSD. The $w1$'s average latency misses the 30μs SLA.

tween hardware and virtualized devices with dynamic queue scheduling. LPNS introduces a fine-grained I/O command scheduling to throttle the competing throughput-intensive workloads to constrain the latency of latency-sensitive workloads. Moreover, we have pioneered the introduction of deterministic network calculus [5, 38] into LPNS to provide mathematical modeling for the latency predictability of virtualized storage, giving a definite latency bound by abstracting the arrival/service curves for storage systems. The experiments show that LPNS can guarantee the latency-predictable QoS and achieve up to **11.57/18.72**$\times$ latency optimization over the SPDK/SR-IOV by eliminating the device-level interference. LPNS can also achieve better latency bounds (50μs) than the state-of-the-art mechanism K2 [48] (80μs) for real-world I/O trace on P5800X and increase up to **1.45**$\times$ additional total throughput over K2 (**1.61GB/s**, equivalent to **47.66%** of the maximum throughput of a P5800X SSD).

To sum up, we make the following contributions:

(1) We analyze the device-level latency interference of the commercially-available but unpredictable NVMe SSDs, and we argue the significance of overcoming this interference from the OS-level storage virtualization design aspects.

(2) We design LPNS, the first OS-level NVMe virtualization solution with latency-predictable QoS enhancement for unpredictable NVMe SSDs in clouds. LPNS designs a self-feedback mechanism that adaptively provides predictable latency according to the workload distributions of multi-tenant VMs. LPNS involves deterministic network calculus to verify the latency upper bound.

(3) We implement LPNS based on mediated pass-through to enhance the original Linux NVMe driver in providing latency-predictable QoS for hybrid-deployed local virtualized and cloud-native storage. All the CSPs can directly use the OS-level LPNS to provide latency-predictable NVMe storage virtualization and sharing without hardware modification and purchase costs.

(4) The evaluations prove the effectiveness of the latency-predictable QoS control of LPNS, compared with previous storage virtualization and other latency QoS control solutions.

The rest of the paper is organized as follows: Section 2 introduces the technical backgrounds of NVMe, cloud storage, and network calculus. Section 3 introduces the motivation for designing scalable and latency-predicable cloud storage virtualization. Section 4 describes the design and implementation of LPNS. Section 5 demonstrates the evaluation results of LPNS. Section 6 introduces the related works, and Section 7 concludes this paper.

2 Background and Motivation

2.1 NVMe Storage

The NVMe SSDs are now widely used in public clouds. The NVMe specification [51] is an efficient and scalable interface designed for high-performance SSDs. NVMe supports up to 65,535 I/O queues whose depth can be up to 65,535. Specifically, each queue pair contains a submission queue (SQ) and a completion queue (CQ). During each I/O execution, the host OS stores IO commands into the SQ and rings the doorbell, and the completion messages are placed into the corresponding CQ by the SSD controller. With the high-parallel SQ/CQ interaction between the host and the SSD controller, NVMe SSDs can obtain high throughput and micro-second-level latency advantages over the traditional interfaces [9], wherein both throughput and latency are extremely significant and mutually restrictive QoS targets of the cloud storage systems.

2.2 Local NVMe Storage for Cloud Services

For better performance and resource utilization, public cloud infrastructures usually adopt two types of solutions to manage their millions of NVMe SSDs. One is running cloud instances or workloads directly on the native servers and using the local storage; another is providing efficient data access to a remote storage pool or dedicated storage servers [36, 41, 46].

However, not all cloud services prefer remote storage solutions, for example, Elastic Compute Services (ECS). We infer there are three main reasons: (1) The remote storage performance is influenced not only by the storage system but also by network devices, which introduces an additional bottleneck of latency performance incurred by the network. (2) Remote storage uses expensive network devices, incurring additional purchase costs to cloud infrastructures and finally hurting the interests of the cloud tenants. (3) Cloud tenants may lease the bare-metal servers or services to customize their own distributed computing and storage clusters, which conflicts with the remote storage architectures.

In contrast, the local storage technique route can provide fast and cheap storage for the public clouds with the widely-used storage virtualization [7, 25, 37, 55, 57, 70]. For example, MDev-NVMe [55] proposes a novel I/O queue pass-through of NVMe hardware queue resources to achieve near-native performance for cloud instance storage. Moreover, local storage virtualization is more flexible in providing QoS guarantees for cloud services, especially the latency-sensitive services that suffer the performance unpredictability of network systems. Therefore, in this paper, we aim to provide latency-predictable virtualized storage services by following the local storage virtualization technique route.

2.3 Deterministic Network Calculus

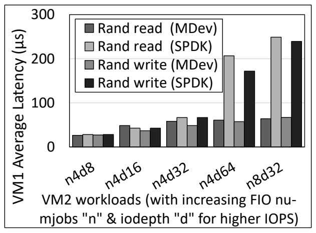
Deterministic Network Calculus (DNC) [38] is used to calculate theoretical worst-case performance guarantees for networks of queues and schedulers, which is commonly used for communication networks to provide predictable latency in typical deterministic queuing systems [13, 40, 59, 65, 69]. The theory's three basic concepts are suitable for providing predictable-latency performance in NVMe virtualization: the arrival curve, the service curve, and the virtual delay. The arrival curve expresses the upper bounds of the number of events that come from the sources over any time. The service curve refers to the guarantee of flows offered by the system and describes the service capability of the system. For a definite system, if the arrival curve and service curve are determined, the virtual delay referring to the delay that an event arriving at a particular time will suffer, can be deduced. The deterministic network calculus proves that the maximum horizontal distance between a workload's arrival curve and service curve is a tight worst-case bound on latency.

According to the NVMe specification and the I/O performance and behavior of the commercial-available NVMe SSDs, we can make a performance assumption that the processing capability of the modern NVMe SSD is stable and constant at most of their working time except during garbage collection and the SSDs serve the I/O command processing in First-In-First-Out (FIFO) policies. Moreover, for the multi-tenant virtual devices sharing the same NVMe SSD, we assume that throughput-intensive workloads can use the maximum queue depth and latency-sensitive processes use a queue depth of 1. The arrival curve refers to the actual IOPS of VM workloads, which determines the commands rate of multiple-tenant workloads that the SSD receives. The service curve guarantees the least command rate that the SSD can process during a busy period. The virtual delay is precisely the I/O latency of the latency-sensitive workload, which is the focus of our attention for latency-predictable QoS in storage virtualization.

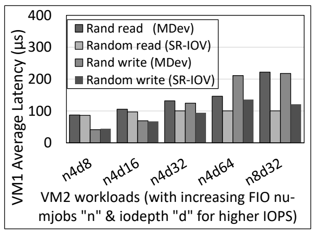
3 Motivation

The clouds aim to provide **latency-predictable Storage QoS** for VMs so their virtual storage devices can have a latency bound for storage I/O operations. However, state-of-the-art storage virtualization [14, 25, 37, 41, 55] cannot solve the device-side latency interference issue and cannot provide latency-predictable QoS. The **device-side latency interference** refers to a phenomenon that the guarantee for the VM with latency-predictable QoS fails when multiple VMs without latency-predictable QoS run throughput-intensive workloads and compete for the same underlying NVMe SSD, incurring the latency deterioration, unbounded latency, and the miss of latency QoS (or SLA) just like the Figure 1 example.

However, even the state-of-the-art NVMe virtualization techniques (including SPDK, SR-IOV, and MDev-NVMe) neglect the importance of eliminating performance interference in multi-tenant storage-sharing scenarios. The more intensive the competitor's workload is, the more severe performance interference happens. We use MDev-NVMe and SPDK vhost-blk to share one Intel Optane P5800X SSD into two virtualized devices. We also use one SR-IOV-capable Samsung PM1735 to build two VFs and use MDev-NVMe as a comparison. In the test cases, VM1 runs a latency-sensitive workload (an FIO [26] random read or write benchmark with *iodepth=1, numjobs=1*), with a growing-intensive 4K random write workload in VM2 by increasing the *numjobs* and *iodepth* parameters of FIO. We separately show the latency performance of the latency-sensitive workload of VM1 in Figure 2, which shows that there is an up to 4.7× latency overhead in MDev-NVMe, 16.5 × overhead in SPDK, and up to 6.1× overhead in SR-IOV over the VM1 workload.



(a) Intel Optane P5800X



(b) Samsung PM1735

Figure 2: Latency interference between two virtualized devices with the state-of-the-art NVMe virtualization. (If VM1 monopolizes one P5800X, the read/write latency of the VM1 workload is 11.9/13.1μs with MDev and 11.9/13.3μs with SPDK. If VM1 monopolizes one PM1735, the read/write latency is 71.9/17.6μs on MDev and 74.9/19.9μs with SR-IOV.)

We further analyze the latency distribution of different I/O phases of NVMe virtualization. We choose MDev-NVMe as a representative to virtualize a P5800X and summarize the results in Table 1. We find that the VM1's average latency on the NVMe controller grows from 62.5% to **93.0%** of the total latency with the increasing IOPS of the VM2 workload. This phenomenon proves that more severe I/O congestion happens when more commands from the competitor workloads simultaneously arrive at the SSD controller and savagely preempt the resources, causing a worse latency bound to the latency-sensitive workloads. Since the hardware/software co-designed virtualization [37, 41] usually attaches standard and unpredictable NVMe SSDs to an accelerator card, we can deduce that these solutions still meet this device-side latency interference and fail to reach latency-predictable QoS.

To overcome the device-level latency interference, we aim to design latency-predictable QoS control for NVMe virtualization. Previous works (summarized in Table 2) usually redesign the Flash Translation Layer (FTL) in the SSD con-

Table 1: VM1 latency distribution in different phases

VM2 IOPS / I/O phase	50K	250K	500K
Guest OS submit commands	2.8%	1.9%	1.1%
Virtual SQ	1.9%	1.9%	0.8%
Physical SQ	1.4%	1.1%	0.4%
SSD controller	**62.5%**	**80.1%**	**93.0%**
Virtual CQ	23.0%	10.2%	2.9%
Virtual Interrupt handling	8.4%	4.8%	1.7%

troller [29,31,53,66] or introduce additional logic into the host software stack [21,48,71] to achieve reliable high-throughput or fair-bandwidth QoS control. FinNVMe [54] local NVMe virtualization designs fine-grained queue-level scheduling to achieve state-of-the-art throughput-oriented QoS control for virtualized devices. For predictable latency, Prioritymeister [74] automatically and proactively configures workload priorities and rate limits to provide tail Latency QoS for shared networked storage. K2 [48] uses work-constraining scheduling to trade reduced throughput for lower latency bound, which is the state-of-the-art latency QoS control among the previous solutions [15,22,29,30,33,36,50,61,64] for native storage. However, Prioritymeister and K2 lack the customized design for NVMe virtualization in multi-tenant cloud storage systems. Moreover, K2 sacrifices too much throughput (up to **2.27GB/s,** equivalent to **47.66%** of the maximum throughput of the P5800X SSD in Section 5 experiments) when reaching predictable latency.

Table 2: Storage resource sharing and scheduling systems.

Systems	**Virtualization Optimized**	**QoS Control**	**Predictable Latency**
VirtIO [57], SPDK [25]	PV	✗	✗
MDev-NVMe [55]	MPT	✗	✗
FinNVMe [54]	MPT	✓	✗
WA-BC [29], LeapIO [41], FVM [37]	SR-IOV	✓	✗
AutoSSD [31], FIOS [53], FLIN [66]	N/A	✓	✗
K2 [48]	N/A	✓	✓
MQFQ [21], D2FQ [71]	N/A	✓	✗
LPNS (Our work)	MPT	✓	✓

*PV: Para-Virtualization. MPT: Mediated Pass-through.

4 LPNS Design and Implementation

4.1 System Overview

Motivated by the analysis of device-side latency interference, we aim to provide predictable latency and overcome the interference problems from the aspect of the OS-level NVMe virtualization design. The QoS levels of different VMs should be determined at initialization and can only be changed when storage service tenants agree. Since the strict predictability generalizing the notion of isolation comes at the overall throughput expense, we should give an upper bound to the latency of NVMe virtualization under definite system settings with a slight total throughput loss of the SSD.

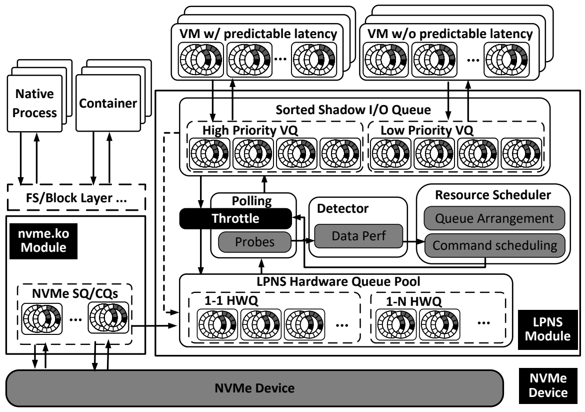


Figure 3: The system architecture of LPNS.

Scalable architecture. We briefly introduce the LPNS architecture in Figure 3. LPNS is designed based on mediated pass-through [27], which has been proven to be excellent in both performance and scalability [54,55,67]. LPNS is implemented as a kernel module to provide virtualized storage with full NVMe features to guest VMs, and it can coexist and cooperate with the original *nvme.ko* module (the NVMe driver) to support the **hybrid deployment of host processes, containers, VMs on each single NVMe SSD**. So it is cheaper, more flexible, and user-friendly for cloud vendors to use LPNS than SR-IOV-capable SSDs or the hardware/software co-designed solutions. Specifically, LPNS designs a performance detector, a queue scheduler, and a command scheduler for predictable latency enhancement, and it provides a flexible polling mechanism for better virtualization scalability.

Full virtualization. LPNS inherits the advantages of physical I/O queues pass-through and active I/O polling from the previous NVMe mediated pass-through solution [55]. LPNS supports full virtualization and does not modify the guest drivers. In the hypervisor (kernel module), the hardware I/O queues (HWQs) can be directly passed-through to VMs, so I/O commands from VMs can be stored in the HWQs through fast I/O paths. The hypervisor maintains a virtual IOMMU structure in shared memory for translating the GPA (Guest Physical Address) of different virtual devices into the IOVA (IO virtual address) of the underlying SSD. Cloud vendors can create partitions and bind each partition with a virtualized storage device with the hypervisor, and the hypervisor can easily do LBA translation between guest and host OS based on the partition information.

Self-feedback QoS control. LPNS has the ability to distinguish QoS targets of VMs to provide latency-predictable QoS. The hypervisor gives each virtual storage a tag when creating the VM to recognize if the workload from this VM should be provided with a predictable latency guarantee [1]. LPNS can periodically trigger resource scheduling between VMs based on runtime performance detection. We place a

[1]We use SVM to represent the VM with latency-predictable QoS guarantees, and use IVM to represent the VM without latency-predictable QoS.

performance probe in the polling thread to periodically collect the index, submission, and completion timestamps of all virtual I/O commands, along with real-time submission and completion command counts. With these statistics, LPNS can calculate the primary execution time of every command and the average latency within a time interval for the scheduler to enhance predictable latency and build an entire self-feedback QoS control system. Specifically, we choose 10 ms as a monitoring interval and 200 ms as a scheduling period in the current implementation.

Flexible and scalable polling. LPNS uses polling threads to process I/O commands and poll the virtual SQ (VSQ) tail, the HWQs, and the virtual CQ (VCQ) head of all VMs for better I/O performance. To reach a balance between performance optimization and CPU overhead, LPNS can arrange only one polling thread for all IVMs, and the thread can fully utilize the high throughput ability of the NVMe SSD; or arrange one dedicated polling thread for each SVM to reach a predictable latency (performance discussed in §4.5). All the polling threads are adaptively triggered on or off according to runtime workload detection of the VMs to reduce unnecessary CPU overhead.

Hardware queue pools (§4.2.) LPNS designs an I/O queue scheduling mechanism with a dynamic queue allocation to achieve a flexible mapping and multiplexing of hardware I/O queue resources. All HWQs are organized into a queue pool as two types of queues, wherein the 1-1 HWQs are exclusive queues for one single VM, and the 1-N HWQs are the shared queues for multiple VMs. The hypervisor implements an HWQ scheduler as a global controller for the HWQ resources. The queue scheduler can periodically schedule the 1-N HWQs between VMs for better virtualization scalability.

I/O command throttling (§4.3.) We implement a virtual I/O command throttling mechanism in the LPNS hypervisor to control the I/O path of each VM and eliminate the device-level latency interference at the OS level. The polling threads can perform the throttling between VMs from the global view. Specifically, the hypervisor provides an interface to adjust the threshold for command throttling, which can control the I/O rate received by the hardware at each scheduling period to reach predictable latency guarantees. The I/O command throttling follows the constraint of deterministic network calculus.

4.2 Scalable I/O Queue Handling

Since previous NVMe virtualization solutions usually use static I/O queue shadowing between virtualized devices and the hardware SSD, the total number of HWQs exposed by the SSD will limit the maximum number of VMs sharing the same underlying SSD. To increase the virtualization scalability, LPNS supports the flexible remapping between HWQs and virtual queues (VQ). Specifically, LPNS can allocate any number of the HWQs (but less than the maximum number of HWQs exposed by the SSD controller) from the *nvme.ko* kernel module into a **Hardware Queue Pool** for I/O queue scheduling, and the rest HWQs can be used by the native applications and containers.

We design an **I/O queue scheduler** to manage the Time Division Multiplexing (TDM) [16] of the HWQs in the **Hardware Queue Pool**. We abstract the HWQs of the pool into two types: 1-1 HWQs and 1-N HWQs. An 1-1 HWQ refers to an HWQ that can only be bound to one VQ. The 1-N HWQs refer to the I/O queues to maintain the necessary I/O capabilities for the other multiple VQs. The total number of 1-1 and 1-N HWQs should be configured when the host system initializes the LPNS module. Specifically, the configuration will not directly change the priority of these HWQs, so it can still work when using the NVMe Weighted-Round-Robin-with-urgent-priority (WRR) feature of the HWQs.

When multiple SVMs and IVMs share the same underlying SSD, LPNS only assign 1-1 HWQs to the SVMs for better latency performance, so the number of 1-1 HWQs should not be less than the number of total VQs owned by all the SVMs. The queue scheduler can schedule the idle 1-1 HWQs and all the 1-N HWQs among the other IVMs.

Since LPNS can monitor and collect real-time performance and workload data of VMs in each period, the I/O queue scheduler follows a hierarchical **workload-aware HWQ scheduling** policy wherein it takes the QoS target and runtime workloads of VMs as the algorithm inputs. The scheduling mainly consists of a hierarchical VQ weight calculating phase and an HWQ switching phase. During each scheduling period, the scheduler first respectively calculates the weight of VQs of all VMs. For any SVM VQ, the weight is set as 0 or the top weight, depending on if this VQ is empty or not. For the VQs of IVMs, their weights are equal to the number of their backlogged commands so that VQs with heavier workloads can get higher priority to be drained quickly. The HWQ switching phase works after the weight updating phase. It sorts the weights of all VQs in descending order, and those high-priority VQs will first use 1-1 HWQs, and the low-priority VQs will be bound to 1-N queues. After switching, the I/O queue scheduler will sleep until the next period.

When one 1-1 HWQ needs to switch to a new VQ, it may still have unfinished commands from the former VQ. Direct forwarding of these stranded commands in VSQs will cause an I/O error because the completion message cannot be handled correctly. So we design a seamless switching mechanism in the I/O queue scheduler. Specifically, LPNS extends the virtual NVMe command structure with an index of the VM to enable the HWQs to interact with different VQs from different VMs simultaneously in the seamless switching. When we want to unbind a 1-1 HWQ from a VQ, the VQ should be bound to another backup 1-N HWQ before the commands in the 1-1 HWQ are executed by the SSD. During the switching, both the 1-1 HWQ and the backup 1-N HWQ can write back the completion information into the original VCQs. Specifically, the VCQ should be locked to ensure data consistency when two HWQs access the same VCQ simultaneously. The seamless switching operations run in the background, and each VQ can continuously send commands to an HWQ without the perception of queue scheduling operations until a new HWQ-VQ binding relationship is established. So it will not disturb the high-parallel fetch of I/O commands from VMs.

4.3 I/O Command Throttling

We design a fine-grained I/O command throttling in LPNS and involve a deterministic network calculus to provide a latency bound of the I/O command for multi-tenant shared virtualized storage. Network calculus has been proven effective in providing latency control for shared network storage systems in previous research, such as Prioritymeister [74]. In our LPNS, we use deterministic network calculus to help to solve the intensive resource and performance competition for different virtual devices sharing the same SSD so that a storage hypervisor can directly eliminate the device-level latency interference from the OS and virtualization layer by scheduling queue resources between different virtual devices and control the I/O command throttling inside a kernel module.

The deterministic network calculus gives a definite bound to the command latency by abstracting the arrival and service curves for storage systems. We can fix the I/O block size to the most widely-used 4K for simplification of the predictable latency deduction. The arrival curve expresses the upper bounds of the number of events from the sources over any time. It refers to the I/O command submission rate by all VMs in our system. The service curve refers to the guarantee of flows offered by LPNS wherein it describes the I/O capability of the NVMe devices. The service curve may be modified by internal functions (like garbage collection and block relocation) of SSDs, so we simplify the mathematical model of LPNS by concentrating on the normal working time without triggering internal functions. And we believe LPNS can cooperate with some future SSDs, such as AutoSSD [31], that try to control the tail latency inside the SSD controller so that LPNS can provide a more robust latency-predictable storage virtualization on the future NVMe SSDs. Once the arrival and service curve of an NVMe SSD is determined, we can deduce the virtual delay at a particular time t, which is the latency bound of SVMs in LPNS virtualization.

Arrival Curve. If the total command submission rate of an intensive VM exceeds θ times the slowest submission rate of a VM with predictable latency QoS guarantee, its I/O will be suspended. The polling threads in the hypervisor of LPNS can trigger the suspension based on the command count of each VM recorded by the performance detector. Using this I/O command throttling threshold is incredibly effective in guaranteeing the latency stability of the VMs with predictable latency guarantees because we can control the command submission rate precisely as we expect. Suppose there are j VMs with intensive workloads co-running with i VMs running latency-sensitive workloads whose IOPS is p and its command queue depth d , the real-time total command submission rate sent to the SSD hardware is:

$$v = p \cdot (j \cdot \theta + i), \tag{1}$$

where the I/O command submission rate v of LPNS is proportional to the IOPS of latency-sensitive workloads.

Similarly, we can get the number of commands b that the sources can send at one time:

$$b = d \cdot (j \cdot \theta + i). \tag{2}$$

Then we formulate the arrival curve of LPNS as:

$$\alpha(t) = v \cdot t + b = (p \cdot (j \cdot \theta + i)) \cdot t + d \cdot (j \cdot \theta + i). \tag{3}$$

Service Curve. The service curve is straightforward because the processing capability of the NVMe SSD is constant (according to the types of NVMe SSDs). We use R to represent the parallel speed of which the hardware processes random write commands per second (since most SSDs has lower random write performance than random read), and L_h to represent the minimum completion latency of an I/O command. So the service curve is:

$$\beta(t) = R \cdot t + L_h. \tag{4}$$

Latency Upper Bound. Call $\Delta(t) = \inf\{\tau \geq 0 : \alpha(t) \leq \beta(t+\tau)\}$. Let $h(\alpha,\beta)$ be the supremum of all values of $\Delta(t)$, then the virtual delay for all t satisfies: $L(t) <= h(\alpha,\beta)$. Given the arrival curve and service curve of LPNS above, we deduce the upper bound of latency $L_{\max}$ as (according to [5, 38]):

$$L_{\max} \leqslant b/R + L_h = d \cdot (j \cdot \theta + i)/R + L_h, \tag{5}$$

where we let $\Omega = j \cdot \theta + i$ to control the predictable latency performance of a latency-sensitive workload p and the total submission rate (which can finally decide the total throughput) on each type of the NVMe SSDs. Specifically, LPNS can adaptively change the θ parameter with the numbers of VMs (i and j) when the performance detector checks if each VM generates active I/O operations when the Ω is determined.

The choice of threshold Ω is essential to the effect of throttling in the real-world system. A smaller Ω value can provide better predictability, but it restricts the throughput of the VMs running throughput-intensive workloads without predictable latency requirements. So the most proper alternative of Ω should be figured out by the optimal upper limit of the hardware processing speed, which needs to be specified and updated by the system administrator because different NVMe SSD may have various R/W performances. Moreover, the requirements for the latency QoS level of each latency-sensitive workload can be stricter and looser in practice, so as the constraints for Ω. Therefore, LPNS can let VMs introduce a tenant-defined latency target in advance and help to tune the Ω parameter. And the decision-maker can increase or decrease Ω by comparing the detected latency with the target during the running time for a better trade-off between throughput and predictable latency. In the evaluation section, we will verify the effectiveness of this latency bound through adequate experiments on different types of NVMe SSDs.

4.4 Latency-Predictable I/O Processing

We use Figure 4 to represent how LPNS enhances latency-predictable I/O processing of the SVMs. The left part of this figure shows the I/O path of a VM with the latency-predictable QoS (the SVM). The LPNS module maintains the shadow I/O queue data structure (directly corresponding to the virtual queue) for each virtual storage in the shared memory between the host kernel and QEMU, which can store I/O commands from guest SQs and generate completion messages into the guest CQs. When guest applications generate I/O operation on the virtualized storage (❶), the command will be stored in the shadow I/O queue. The polling thread will immediately poll the head of the queue (❷), and translate the DMA and LBA addresses in commands, store it into the 1-1 HWQ, and finally ring the hardware doorbell register. (❸) - (❺) represents the process that the SSD controller fetches command, generates DMA, and stores the completion messages into HWQs. The polling thread will continuously check if the doorbell register of the hardware CQs updates to accelerate the I/O operation instead of waiting for the SSD controller to inject interrupts (❻). And the polling thread will compose the completion message of the VMs and store the message into the shadow queue (❼), and inject a virtual interrupt into the VM. Finally, the guest VM driver can complete the I/O operation (❽).

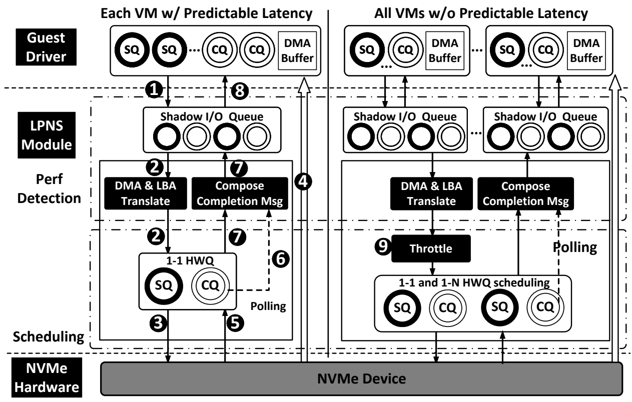
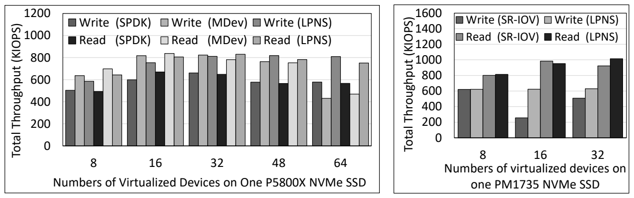


Figure 4: I/O work flow of LPNS virtualized storage.

The right part of Figure 4 shows that the IVMs get the HWQ resources from HWQ scheduling, and the polling thread can throttle the command distribution from the VQ to HWQ (where ❾ replaces the ❷). With the I/O path designs, the intensive workloads in these VMs will not hurt the latency of the VM with the predictable latency guarantees because of the device-level latency interference.

4.5 Discussion

Polling effectiveness. We discuss the polling effectiveness of LPNS by comparing the throughput of LPNS with MDev-NVMe, SPDK, and SR-IOV when multiple VMs share a P5800X/PM1735 SSD. Figure 5 demonstrates the total throughput where increasing numbers of VMs running an FIO workload with the "*numjobs*=1, *iodepth*=1" parameters share the same NVMe SSD. The results prove that LPNS can fully utilize the P5800X or PM1735 for multiple IVMs with only one polling thread and provide better scalability than MDev-NVMe, SPDK, and SR-IOV.



(a) Intel Optane P5800X (b) Samsung PM1735

Figure 5: Polling effectiveness for high throughput (SPDK and LPNS both use one polling thread shared by all VMs.)

We also run an FIO test with "*numjobs*=1 or 4, *iodepth*=1" on the host OS and inside a VM with MDev-NVMe, SPDK and SR-IOV. Figure 6 demonstrates how one dedicated polling thread can achieve promising and near-native average latency.

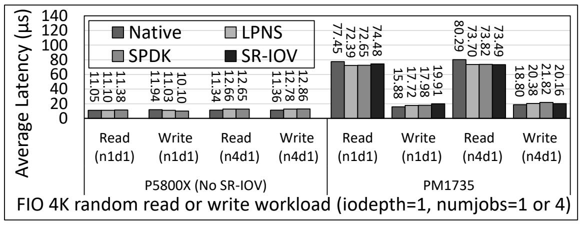


Figure 6: One dedicated polling thread for one SVM.

Performance overhead. In the self-feedback QoS control, each polling thread uses a two-phase array to achieve real-time performance data feedback for the SVMs. While the thread reads data from one phase of the array for computation, the probes record current I/O data into the other phase. It is lock-free, and only a writing-array operation is added into the origin command execution time. So the performance overhead of our self-feedback mechanism is negligible. Figure 5 and 6 prove that the active polling in LPNS can ensure no utilization overhead of the total IOPS and near-native idle latency performance for SVMs.

Resource overhead. (1) The choices of the performance detection are critical for resource overhead. The detection exacerbates kernel memory consumption because tens of thousand of commands can occupy hundreds of KBs of memory to store the performance data. Therefore, we cut the one-period detection into 10 ms intervals (much shorter than the scheduling period), we can calculate the average command latency for each interval, and finally sum up the results of a whole period so we can reduce kernel memory overhead. (2) The polling threads in LPNS can be adaptively turned into an idle status when there are no I/O operations generated from the guest machines in a recent 500 ms. When performance detection finds there are burst I/O operations, LPNS will turn on the polling thread immediately. This helps reduce the CPU overhead of polling.

Limitation. The deterministic network calculus of LPNS needs tunning when cloud workloads use different NVMe SSDs or block sizes instead of the most commonly-used 4K. Also, for the SSDs whose random write IOPS is much lower than random read, the deterministic network calculus will be more strict to provide latency bound, and LPNS may sacrifice more throughput performance than when running LPNS on SSDs with equivalent random read and write performance, such as Intel Optane NVMe SSDs.

5 Evaluation

In this section, we evaluate LPNS on different NVMe SSDs and compare LPNS with several famous related works.

Firstly, we want to compare LPNS with mainstream NVMe virtualization solutions. We start with the following NVMe virtualization mechanisms: MDev-NVMe, VirtIO, SPDK with *vhost-blk* interfaces, and SR-IOV [2].

We also compare LPNS with state-of-the-art QoS control systems, MQFQ, D2FQ, and K2. Specifically, (1) we build a K2 kernel module with its source code [68]; (2) We implement MQFQ with 1125 Lines of Code according to the description in [21] since the original source [20] is no longer accessible. (3) We modify 19 lines of the D2FQ source codes [62] for bug fixing according to the description in [71].

5.1 Experiment Setup

Hardware configuration. We evaluate LPNS on two servers. One server has two 20-core Intel Xeon Gold 6248 CPUs (2.5GHz), 384GB DDR4 memory, and one 400GB Optane P5800X SSD [24]. Another server has two 20-core Intel Xeon Gold 6230 CPUs (2.1GHz), 128GB DDR4 memory, and one 1.6TB SR-IOV-capable Samsung PM1735 SSD [58]. The parameters Ω we choose for the P5800X and PM1735 are 190 and 100 (the choices are discussed in §5.5.)

System configuration. We implement LPNS based on Linux kernel 5.0.0. The two host servers run a Ubuntu 18.04.3 LTS 64bit OS and boot VMs with the same OS image version based on KVM/QEMU. There are different numbers of SVMs and IVMs in micro and real-world benchmarks. Each VM has 4 VCPUs, 4GB memory, 40GB virtual NVMe storage, and 4VQs equal to the number of VCPUs. Each virtual storage is created on a logical partition of the SSD, and the VM uses the original NVMe driver of Linux. In all experiments, the total number of HWQs used for virtualization is less than the total number of VQs to mimic a resource shortage scenario in the real-world cloud environments.

[2]The hardware/software co-designed FVM [37] and LeapIO [41] are not open-sourced and available.

Workload configuration. The micro-workloads are generated by FIO [26], which is widely used in both industry and research. The FIO version is 3.1, and *libaio* is selected as the default I/O engine. We set the I/O mode as Direct I/O. We set the block size of random read/write as 4K. In application benchmarks, we first replay the webuser service of open-sourced production systems at Florida International University (FIU) [6]. We also use YCSB [8] as another application benchmark and choose RocksDB [3] to test the I/O performance of K-V store. The YCSB version is 0.17.0, and we use the embedded RocksDB database of YCSB [2].

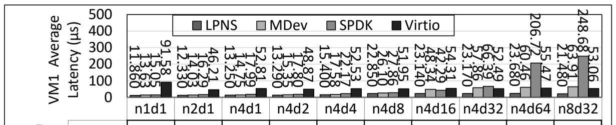
5.2 Micro Benchmarks

In the micro benchmarks, we let one SVM to share a P5800X SSD with one or multiple IVMs on LPNS, MDev-NVMe, SPDK with *vhost-blk*, and VirtIO. We also compare LPNS with MDev-NVMe and SR-IOV on a PM1735. In each test case, the SVM runs the lightest FIO workloads in a general sense by setting both the two standard decisive FIO parameters “numjobs" and “iodepth" as 1 (n1d1). The other IVMs run throughput-insensitive workloads as competitors, and we let their workloads grow from as light as the SVM to heavy enough to reach the throughput limit of the SSD by increasing the “numjobs" and “iodepth" parameters. We observe the latency of the “n1d1" workload in the SVM when the SVM faces serious interference.

The micro benchmark results are demonstrated in Figure 7. Firstly, we let one SVM (VM1) and IVM (VM2) to share one underlying P5800X or PM1735 SSD and show the performance results in Figure 7a to 7d. The bar charts in the upper part of the sub-figures represent the latency performance of the FIO n1d1 workload in VM1, and the tables in the lower part are the throughput of the competitor workloads of VM2 in the same test cases. We also let one SVM (VM1) and multiple (2-7) IVMs to share the same SSD and demonstrate the latency of the VM1 workload in Figure 7e.

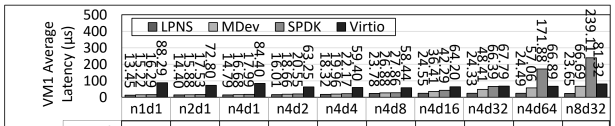
P5800X. We let the VM1 and the VM2 share one P5800X SSD, and the performance of 4K random read and write test cases are depicted in Figure 7a and 7b. Optane SSDs use 3D XPoint technology [17] (the most advanced storage medium), and their hardware controllers usually have stable storage service capability in latency and throughput, which is more friendly to latency-predictable systems. Because Optane SSDs also have similar random read and write performance, the results shown in Figure 7a and 7b have similar features.

LPNS can bound the latency of VM1 workload at a low level (less than 25 μs) in both 4K random read and write cases. LPNS can only sacrifice the throughput of VM2 workload within 7.0% of MDev-NVMe and within 8.2% of SPDK when VM2 runs n464 or n832 cases, and in most cases, the sacrifices are less than 1%. Moreover, LPNS successfully eliminates the device-level interference observed in Figure 2.



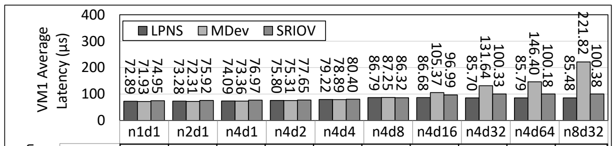
	n1d1	n2d1	n4d1	n4d2	n4d4	n4d8	n4d16	n4d32	n4d64	n8d32
LPNS	249	480	902	1410	2045	2194	2186	2152	2168	2179
MDev	253	484	902	1412	2053	2257	2311	2315	2317	2323
SPDK	209	412	764	1203	1796	2198	2290	2339	2361	2368
Virtio	72.8	89.4	287	424	548	584	577	594	541	560

(a) One SVM & One IVM, 4K random read (P5800X).



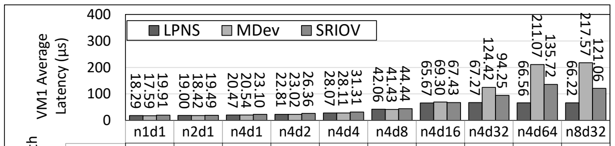
	n1d1	n2d1	n4d1	n4d2	n4d4	n4d8	n4d16	n4d32	n4d64	n8d32
LPNS	216	406	747	1175	1673	2064	2004	2002	2048	2088
MDev	210	402	767	1193	1701	2117	2139	2187	2191	2208
SPDK	186	357	682	1016	1510	1916	2073	2162	2200	2197
Virtio	37.9	108	251	413	517	512	587	542	509	441

(b) One SVM & One IVM, 4K random write (P5800X).



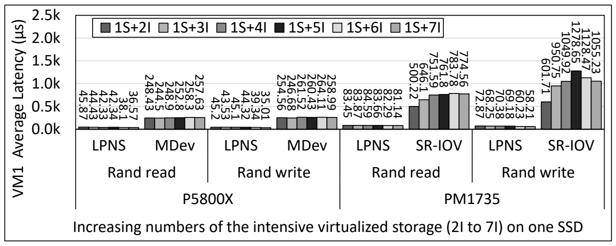
	n1d1	n2d1	n4d1	n4d2	n4d4	n4d8	n4d16	n4d32	n4d64	n8d32
LPNS	54.3	198	214	417	779	1371	1797	1772	1756	1760
MDev	56.6	111	219	423	791	1386	2177	2736	2787	2562
SRIOV	63.1	125	261	509	961	1699	2490	2539	2549	2547

(c) One SVM & One IVM, 4K random read (PM1735).



	n1d1	n2d1	n4d1	n4d2	n4d4	n4d8	n4d16	n4d32	n4d64	n8d32
LPNS	213	394	718	1131	1702	2108	2148	2148	2154	2100
MDev	225	432	783	1243	1819	2128	2237	2259	2273	2214
SRIOV	192	389	765	1217	1766	2162	2204	2223	2235	2175

(d) One SVM & One IVM, 4K random write (PM1735).



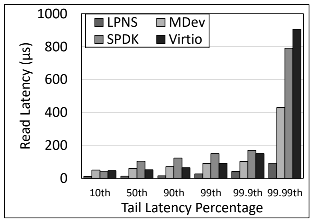
(e) One SVM & Multiple IVMs (P5800X & PM1735)

Figure 7: The average latency of LPNS compared with the other NVMe virtualization on P5800X and PM1735.

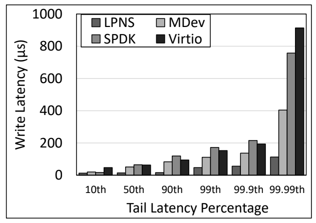
In MDev-NVMe and SPDK, the increasing throughput of the competitor workload in VM2 will lead to latency deterioration of the VM1 workload. For example, when using n4d64 or n8d32 parameters, LPNS can reach up to **11.57/2.98**× latency optimization over SPDK/MDev-NVMe. The main reason is that their polling threads cannot provide strong performance isolation between different virtualized devices to overcome latency interference. VirtIO can obtain stable average latency and better latency bound for VM1 than SPDK, but the VM1 latency and VM2 throughput are not comparable with LPNS because VirtIO is generic virtualization for block devices and not optimized for NVMe.

Figure 7e shows that LPNS can provide latency-predictable QoS with promising scalability for the SVM when we increase the number of IVMs from 2 to 7. The latency results of VM1 are very stable and low, and bounded by **50** μs, which can achieve up to 7.40× latency optimization of MDev-NVMe.

Figure 8 demonstrate the tail latency of the n8d32 test cases in Figure 7a and 7b. LPNS can reach up to 3.84/6.70× optimization of 99.9th/99.99th 4K random read tail latency of SPDK and 3.73/9.96× optimization of 99.9th/99.99th 4K random write tail latency of SPDK.



(a) Random Read Tail Latency



(b) Random Write Tail Latency

Figure 8: Tail Latency of micro benchmarks on P5800X .

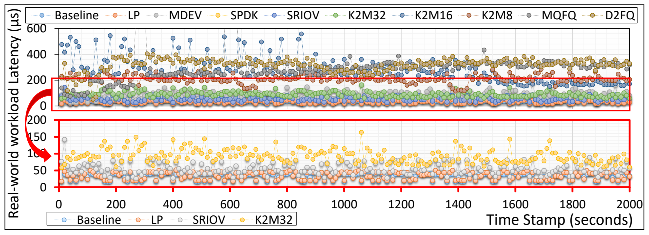
PM1735. We let the VM1 and the VM2 share one PM1735 SSD, and the performance results are shown in Figure 7c and 7d. These results show that LPNS can provide better latency than MDev-NVMe and SR-IOV in all cases and achieve up to 3.27× latency optimization. Moreover, when we increase the number of IVMs from 2 to 7 on PM1735, the IVMs cause serious latency interference on the VM1 when using SR-IOV. LPNS can bound the VM1 workload's latency under **90**μs, with up to **18.72**× latency optimization over SR-IOV. To be mentioned, the throughput performance loss of IVMs on PM1735 is worse than P5800X, which is less than 7.07% in the 4K random write cases, but up to 31.11% in the 4K random read cases. The main reason is that the random read service capability of the PM1735 controller is much better than the random write, and LPNS must choose a more conservative configuration to ensure latency-predictable QoS but incur more throughput loss.

In general, LPNS can perform better in both latency bound and low throughput loss than VirtIO. LPNS can provide ultra-low latency and ensure latency-predictable QoS compared to MDev-NVMe, SPDK, and SR-IOV.

5.3 Real-world I/O trace Replay

In the real-world I/O trace replay, we run a *webuser* workload in the SVM(s) and let the SVM share one Optane P5800X and PM1735 SSD with one or multiple IVMs running intensive workloads (FIO 4K random write with *numjobs=4* and *iodepth=32*). The raw data of the latency-sensitive *webuser* server comes from [6]. We extract the throughput and the proportion of reading and writing operations from the raw data.

The running time of each *webuser* test case is 2000 seconds, and we report latency results every 10 seconds.


(a) Samsung PM1735.

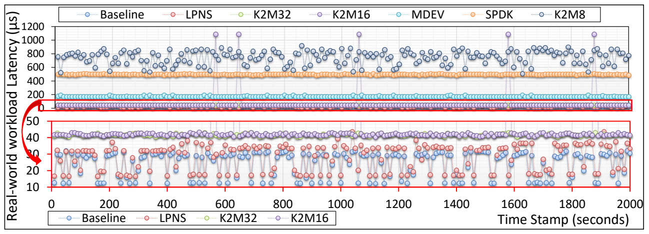

(b) Intel Optane P5800X.

Figure 9: The latency of real-workload *webuser* on LPNS, MDev-NVMe, SPDK, MQFQ, D2FQ, and K2 with 32, 16, and 8 *max_inflight* when 4 VMs share one NVMe SSD (VM1 is an SVM and VM2-4 are IVMs), compared with a baseline that only VM1 monopolizes the NVMe SSD. The lower-part charts in the figures are the enlargement of the data from the red boxes of the upper-part charts.

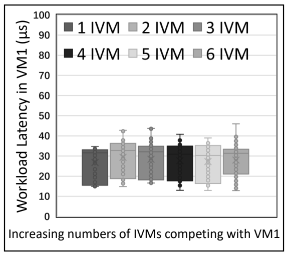
We choose MDev-NVMe and SPDK with *vhost-blk* as the representative of the state-of-the-art virtualization, and we design eight groups of experiments with different numbers of SVM (s) and IVM(s), including "1SVM & 1IVM" to "1SVM & 6IVM", "2SVM & 2IVM", and "3SVM & 1IVM".

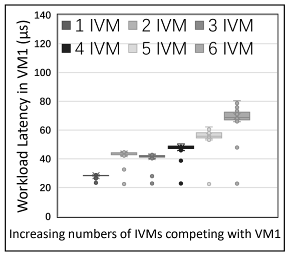
We choose K2, MQFQ, and D2FQ for comparison. Since LPNS, MDev-NVMe, and SPDK can prove near-native latency for each virtualized storage and K2, MQFQ, and D2FQ are not designed for NVMe virtualization, we deduce that this latency performance comparison are typical and persuasive. We build the K2 module and *insmod k2-scheduler* into the original 4.15.0-175-generic kernel of the public Ubuntu 18.04 system. We implement MQFQ and D2FQ in a Linux 5.3.10 kernel. We run the same real-world *webuser* workloads and intensive FIO random write workloads on the host OS, just like inside the SVM(s) and IVM(s). We use the `ionice` [10] commands to give the *webuser* a high priority for latency QoS control on K2, MQFQ, and D2FQ scheduler. Specifically, we configure the *max_inflight* parameters of K2 as 32, 16, and 8, and the parameters of MQFQ and D2FQ are set as default values according to the papers.

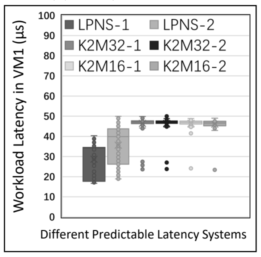
Latency fluctuation. We firstly choose the "1 SVM & 3 IVMs" cases on PM1735 and P5800X as a typical example of all the cases to show the overall latency control effects of the related works compared with a baseline where only VM1 running the *webuser* monopolizes the entire SSD. We demonstrate the latency results of the *webuser* in Figure 9.

On both PM1735 and P5800X SSD, the latency of LPNS can nearly coincide with the baseline fluctuation line, and it successfully ensures the *webuser*'s latency-predictable QoS. SR-IOV and K2 can also provide better latency performance than MQFQ, D2FQ, and SPDK because the main purpose of MQFQ and D2FQ is to reach a fair queue scheduling and SPDK does not involve reliable performance isolation. On P5800X, K2 with *max_inflight=32* can successfully bound the latency within **50**μs. However, SR-IOV and K2 cannot bound latency within 50μs level as LPNS does on PM1735.

Latency interference elimination. We count the latency distributions of the eight groups of LPNS and K2 (providing better latency bound effect in the example case) in Figure 10 with box charts.


(a) LP (1 S-VM + n I-VM)


(b) K2M32 (1 S-VM + n I-VM)


(c) 2 S-VM+2 I-VM

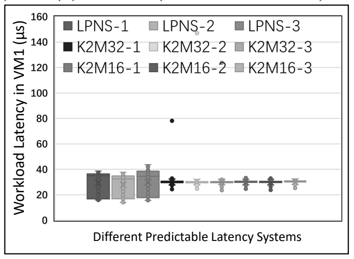

(d) 3 S-VM+1 I-VM

Figure 10: The latency distribution results of test cases where different numbers of SVMs and IVMs share one P5800X SSD on the LPNS system or on a system using K2.

In general, LPNS can bound the latency by 50 μs in all the test cases. K2 can bound the latency by 50 μs when there are small numbers (1∼4) of IVMs. However, the outliers of latency results in the box chart prove that the latency of VM1 *webuser* workload grows over 50 μs on K2 with *max_inflight=32* when there are more than 5 IVM as competitors. When there are more SVM but less IVM (for example, Figure 10c and 10d), K2 with *max_inflight=32 or 16* can effectively bound the latency by 50 μs just as LPNS does.

Throughput sacrifice. Figure 11 demonstrates the total throughput of the IVM(s) that compete for resources with the SVM in the "1SVM & 1IVM" to "1SVM & 6IVM" cases. We choose the maximum throughput of the P5800X as the baseline. LPNS throughput loss to MDev-NVMe is 18.46% in average and less than 19.88%. Figure 11 also proves that LPNS can reach latency-predictable QoS with less throughput loss of the IVMs than K2 when multiple VMs share the one SSD. For example, when there are 6 IVMs (or 6 native intensive workloads), LPNS can increase up to **1.45** × additional total throughput over K2 with *max_inflight=32* (equivalent to **47.66% of the maximum throughput** of the P5800X SSD.)

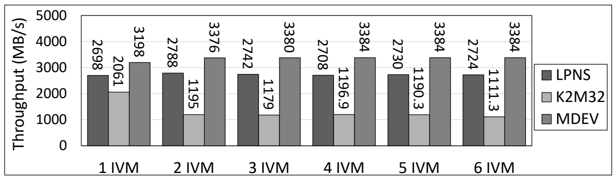


Figure 11: The total throughput of the one to six IVMs (or native workloads) on LPNS and K2 (*max_inflight=32*) in Figure 10a and 10b (MDev-NVMe as the baseline).

We deduce that LPNS will be an excellent choice to achieve their latency-predictable QoS with less sacrifice of the SSD maximum throughput when cloud vendors want to increase the scalability of virtualized devices on each single SSD with heavier over-subscription of storage resources.

5.4 YCSB Key-Value Store on RocksDB

We use the YCSB to generate K-V store workloads on the RocksDB databases in one SVM, and use several IVMs running intensive FIO random read/write workloads to generate serious interference and demonstrate that LPNS can provide latency-predictability for K-V store applications. Specifically, we build an `ext4` [1] file system on the virtualized NVMe storage device in the SVM to run the RocksDB database. We run the generic configuration of YCSB from the `workloada` to `workloadf`, which uses Zipfian [18] distributions. We set up 20M requests on 4GB database by enlarging the "recordcount" and "operationcount".

Figure 12 demonstrates the average latency performance results of YCSB benchmarks on the RocksDB databases, including the baseline where the benchmark monopolizes the entire P5800X SSD, or using MDev to support 1SVM +3IVM, or using LPNS to support 1SVM +3IVM. In the six YCSB tests, LPNS can efficiently reduce the latency interference from the IVMs and provide promising average latency performance for the Read, Update, Insert, or Scan operations. For example, the YCSB-A and YCSB-B workloads are identical mix Read and Update operations, which will bring challenges to the latency QoS control systems. The results prove that LPNS can achieve up to **7.41** × latency optimization of MDev-NVMe in the YCSB-B workloads. In YCSB-C, E, and F, MDev-NVMe and LPNS can provide latency similar to the monopolized baselines. We infer the reasons: YCSB-C is a 100% read case, which is very friendly to the cache systems, so it is not seriously interfered by the device-level congestion; YCSB-E and YCSB-F are more performance-critical about the computation ability of the K-V store databases, so the storage system is not a performance bottleneck in these cases.

In Table 3, we count the tail latency results of YCSB-A and YCSB-B benchmarks of MDev-NVMe and LPNS from Figure 12. From these results, we can find that LPNS can

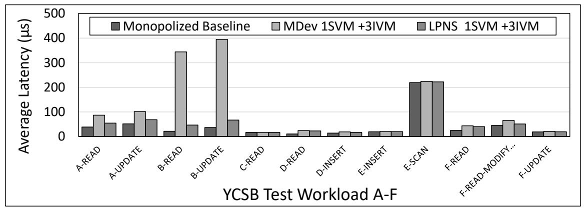


Figure 12: The average latency of YCSB A to F workloads.

achieve up to 4.44 × optimization of the 95th tail latency and up to 4.27 × optimization of the 99th tail latency compared with MDev-NVMe, which can provide the most advanced tail latency among the traditional NVMe virtualization in the Section 5.2 experiments.

Table 3: YCSB tail latency in YCSB-A and YCSB-B

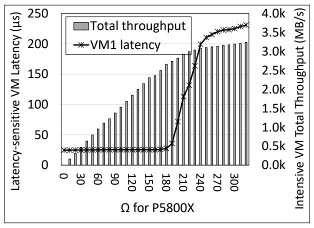
		Read		Update	
		MDev	LPNS	MDev	LPNS
YCSB-A	P95 (us)	419	242	453	260
	P99 (us)	896	366	934	399
YCSB-B	P95 (us)	907	204	961	225
	P99 (us)	969	317	1060	358

With these experimental results, we imply that LPNS can provide latency-predictable QoS for the latency-sensitive applications such as the K-V store databases in cloud services.

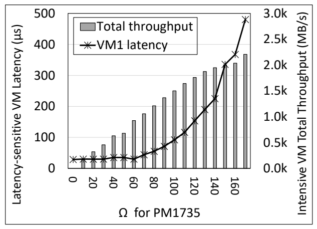
5.5 Predictability Trade-off

We discuss the trade-off between latency and throughput to guide the choice of the proper predicted value, which refers to the value of the I/O command scheduling parameter Ω in LPNS. The Ω value is related with the hardware SSD and can be flexibly determined by the cloud storage administrators to make an optimal alternative according to the hardware and workload scenarios. We observe the latency performance of SVM and the total throughput of IVMs to evaluate the effect of different Ω values and choose the proper Ω.

Figure 13 plots the latency of VM1 (SVM) and total throughput of IVMs at different Ω values on the P5800X and PM1735 SSDs. We use the dot lines to indicate the latency performance and bars to indicate the total throughput performance.



(a) Intel Optane P5800X.



(b) Samsung PM1735.

Figure 13: Performance trade-off for different SSDs.

On P5800X, the R is 800K IOPS (0.8 iopμs) and L_h of the workload is 11.05 μs (measured when there is only one workload that monopolizes the SSD). When we choose Ω as 10, the latency can be bounded within 23.55μs according to Eq. 5, which matches the results in Figure 13a.

However, when Ω is smaller than 160, keeping decreasing Ω can not significantly improve the latency, but the throughput will continue to decline. When Ω is larger than 240, keeping increasing Ω is also uneconomic because the throughput is close to the ceiling while the latency is still growing. So an appropriate Ω value must be between 160 and 240, depending on the trade-off between the latency requirements and the acceptability of bandwidth degradation. We choose **190** on P5800X in all our experiments to bound the latency in micro and application benchmarks and get a balance between the latency bound (**50**μs) and the throughput sacrifice. With these configurations, we use Figure 14 to prove that LPNS can successfully bound the latency of the workload in Figure 1.

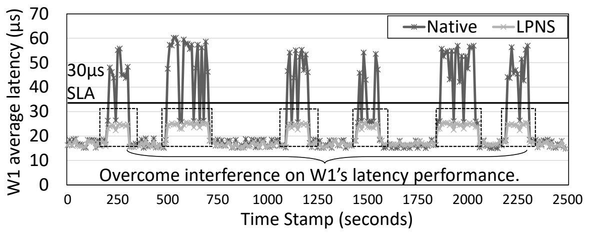


Figure 14: Latency bounded within 30μs without SLA miss.

Compared with P5800X, it is more difficult to reach a promising balance between latency and throughput on the PM1735. The Ω parameter when Ω is smaller than 60, keeping decreasing Ω can not significantly improve the predictable latency but only decline the throughput. And when we choose larger Ω, the latency grows faster than on P5800X. We finally choose **100** as the Ω to provide an upper bound of latency at **100**μs for the latency-sensitive workloads on PM1735. However, it will incur more serious throughput loss than using LPNS on P5800X. And this will be one of our future work to reach a better balance of latency prediction and throughput by upgrading our design and implementation.

6 Related Work

Local NVMe virtualization. The state-of-art local NVMe virtualization mechanisms include para-virtualization *VirtIO* [57], a userspace NVMe driver *VFIO* [70], and SPDK [25], MDev-NVMe [55], FinNVMe [54], FVM [37]. *VirtIO* is an I/O para-virtualization framework, which provides an abstraction of a set of common simulation devices suffers from the poor performance of virtualized devices. Fam Zheng [73] used *VFIO* to implement the NVMe driver to work with the modified user-space NVMe driver in Qemu. The SPDK is a userspace and lockless NVMe driver that provides an efficient and scalable interface to access various storage devices. MDev-NVMe uses mediated pass-through and the active polling mode to achieve high I/O performance. FVM implements a storage virtualization layer on an FPGA card to offload virtualization overhead, and FVM can ensure high throughput but incurs about 25% latency overhead over native performance.

Storage QoS. Many researches concentrate on NVMe resource sharing and scheduling by modifying the device controller or host software stack [4, 11, 19, 32, 39, 42, 43, 56, 60, 72]. FIOS [53] achieves the fairness of resource sharing through the per-task timeslices. AutoSSD [31] employs a self-management mechanism to schedule device-internal background jobs to prevent the SSD from falling into a critical condition that causes long tail latency. PartFTL [49] splits flash storage into separate read and write sets to ensure writes never block reads. FLIN [66] is a lightweight transaction scheduler using a three-stage scheduling algorithm to provide fairness, implemented within the SSD controller firmware. For performance isolation [15, 22, 30, 33, 36, 50, 61, 64] and QoS, workload-aware budget compensation (WA-BC) [29] provides a device-level scheduler with SR-IOV for performance isolation and fairness among multiple VMs by penalizing noisy neighbors. Differentiated Storage Services [47] propose an I/O classification architecture to close the gap between computer systems and storage systems, and improve end-to-end performance, reliability, and security of storage. More recent schedulers have evolved to guarantee isolation at OS-level without modification to the hardware. K2 [48] is a lightweight and device-agnostic I/O scheduler for Linux targeting NVMe-attached storage. Multi-Queue Fair Queueing (MQFQ) [21] is a fair and work-conserving scheduler for multi-queue systems. D2FQ [71] uses the device-side scheduling feature (NVMe WRR) to reach low-CPU-overhead fair queue scheduling.

7 Conclusion

In this paper, we propose LPNS, a latency-predictable NVMe virtualization mechanism for local cloud storage. Based on the mediated pass-through mechanism, LPNS retains high virtualization performance with I/O queue and command scheduling. To prove latency-predictable QoS, we model LPNS as a deterministic queuing system and deduce the latency upper bound referring to the deterministic network calculus. The evaluations demonstrate that LPNS can achieve the goal of latency predictability and prove to be an efficient cloud storage virtualization mechanism. In our future work, we will also improve the balance between latency prediction and throughput on more types of NVMe SSDs, and we are devoted to integrating LPNS into a hardware/software co-designed solution to offload overhead and improve scheduling efficiency.

Acknowledgments

We sincerely thank our shepherd, Youyou Lu, and the anonymous reviewers for their suggestions and help to improve the paper. This work was supported in part by National Key Research & Development Program of China (No. 2022YFB4500103), the Programs for NSFC (No. 62032008), the Resnics Project, and STCSM (No. 20510712400). The corresponding author is Jianguo Yao.

References

[1] Ext4 filesystem. https://www.kernel.org/doc/Documentation/filesystems/ext4.txt/.

[2] how to run ycsb on rocksdb. https://github.com/brianfrankcooper/YCSB/tree/master/rocksdb/.

[3] A persistent key-value store for fast storage environments. https://rocksdb.org/.

[4] Sungyong Ahn, Kwanghyun La, and Jihong Kim. Improving I/O resource sharing of linux cgroup for nvme ssds on multi-core systems. In *Proceedings of the 8th USENIX Workshop on Hot Topics in Storage and File Systems, HotStorage 2016, Denver, CO, USA, June 20-21, 2016*. USENIX Association, 2016.

[5] Marc Boyer, Euriell Le Corronc, and Anne Bouillard. *Deterministic network calculus: From theory to practical implementation*. John Wiley & Sons, 2018.

[6] Camelab. Flash-based block traces. http://trace.camelab.org/2016/03/01/flash.html.

[7] Yiquan Chen, Jiexiong Xu, Chengkun Wei, Yijing Wang, Xin Yuan, Yangming Zhang, Xulin Yu, Yi Chen, Zeke Wang, Shuibing He, et al. Bm-store: A transparent and high-performance local storage architecture for bare-metal clouds enabling large-scale deployment. In *2023 IEEE International Symposium on High-Performance Computer Architecture (HPCA)*, pages 1031–1044. IEEE, 2023.

[8] Brian F Cooper, Adam Silberstein, Erwin Tam, Raghu Ramakrishnan, and Russell Sears. Benchmarking cloud serving systems with ycsb. In *Proceedings of the 1st ACM symposium on Cloud computing*, pages 143–154, 2010.

[9] Intel Cooperation. AHCI specification for Serial ATA. https://www.intel.com/content/www/us/en/io/serial-ata/ahci.html.

[10] die.net. ionice(1) - Linux man page. https://linux.die.net/man/1/ionice.

[11] Adam Ji Dou, Song Lin, and Vana Kalogeraki. Real-time querying of historical data in flash-equipped sensor devices. In *Proceedings of the 29th IEEE Real-Time Systems Symposium, RTSS 2008, Barcelona, Spain, 30 November - 3 December 2008*, pages 335–344. IEEE Computer Society, 2008.

[12] Nathan Farrington and Alexey Andreyev. Facebook's data center network architecture. In *Proceedings of the 2013 Optical Interconnects Conference, OI 2013, Santa Fe, NM, USA, May 5-8, 2013*, pages 49–50, 2013.

[13] Fabien Geyer and Steffen Bondorf. Deeptma: Predicting effective contention models for network calculus using graph neural networks. In *IEEE INFOCOM 2019-IEEE Conference on Computer Communications*, pages 1009–1017. IEEE, 2019.

[14] PCIe Special Interest Group. Welcome to pci-sig. https://pcisig.com/, 2020.

[15] Ajay Gulati, Irfan Ahmad, and Carl A. Waldspurger. PARDA: proportional allocation of resources for distributed storage access. In *Proceedings of the 7th USENIX Conference on File and Storage Technologies, February 24-27, 2009, San Francisco, CA, USA. Proceedings*, pages 85–98. USENIX, 2009.

[16] Ajay Gulati, Arif Merchant, and Peter J. Varman. mclock: Handling throughput variability for hypervisor IO scheduling. In *Proceedings of the 9th USENIX Symposium on Operating Systems Design and Implementation, OSDI 2010, October 4-6, 2010, Vancouver, BC, Canada, Proceedings*, pages 437–450. USENIX Association, 2010.

[17] F. T. Hady, A. Foong, B. Veal, and D. Williams. Platform storage performance with 3d xpoint technology. *Proceedings of the IEEE*, 105(9):1822–1833, 2017.

[18] S Haitun. Stationary scientometric distributions: Part iii. the role of the zipf distribution. *Scientometrics*, 4(3):181–194, 1982.

[19] Mingzhe Hao, Levent Toksoz, Nanqinqin Li, Edward Edberg Halim, Henry Hoffmann, and Haryadi S. Gunawi. LinnOS: Predictability on unpredictable flash storage with a light neural network. In *14th USENIX Symposium on Operating Systems Design and Implementation (OSDI 20)*, pages 173–190. USENIX Association, November 2020.

[20] hedayati. . https://github.com/hedayati/mqfq.

[21] Mohammad Hedayati, Kai Shen, Michael L. Scott, and Mike Marty. Multi-queue fair queuing. In *Proceedings of the 2019 USENIX Annual Technical Conference, USENIX ATC 2019, Renton, WA, USA, July 10-12, 2019*, pages 301–314. USENIX Association, 2019.

[22] Jian Huang, Anirudh Badam, Laura Caulfield, Suman Nath, Sudipta Sengupta, Bikash Sharma, and Moinuddin K. Qureshi. Flashblox: Achieving both performance isolation and uniform lifetime for virtualized ssds. In *Proceedings of the 15th USENIX Conference on File and Storage Technologies, FAST 2017, Santa Clara, CA, USA, February 27 - March 2, 2017*, pages 375–390. USENIX Association, 2017.

[23] Khoa Huynh. Exploiting the latest kvm features for optimized virtualized enterprise storage performance. *CloudOpen, North America*, 2013.

[24] Intel. Intel® Optane™ SSD DC P5800X Series. https://ark.intel.com/content/www/us/en/ark/products/201861/intel-optane-ssd-dc-p5800x-series- 400gb-2-5in-pcie-x4-3d-xpoint.html.

[25] Intel. Storage performance development kit. `http://www.spdk.io/`.

[26] Axboe Jens. Fio: Flexible io tester. `https://github.com/axboe/fio`.

[27] Neo Jia. VFIO Mediated devices. `https://www.kernel.org/doc/Documentation/vfio-mediated-device.txt`.

[28] Yichao Jin, Yonggang Wen, and Qinghua Chen. Energy efficiency and server virtualization in data centers: An empirical investigation. In *Proceedings of the 2012 Proceedings IEEE INFOCOM Workshops, Orlando, FL, USA, March 25-30, 2012*, pages 133–138. IEEE, 2012.

[29] Byunghei Jun and Dongkun Shin. Workload-aware budget compensation scheduling for nvme solid state drives. In *Proceedings of the IEEE Non-Volatile Memory System and Applications Symposium, NVMSA 2015, Hong Kong, China, August 19-21, 2015*, pages 1–6. IEEE, 2015.

[30] Jeong-Uk Kang, Jeeseok Hyun, Hyunjoo Maeng, and Sangyeun Cho. The multi-streamed solid-state drive. In *Proceedings of the 6th USENIX Workshop on Hot Topics in Storage and File Systems, HotStorage '14, Philadelphia, PA, USA, June 17-18, 2014*. USENIX Association, 2014.

[31] Bryan Suk Kim, Hyun Suk Yang, and Sang Lyul Min. Autossd: an autonomic SSD architecture. In *Proceedings of the 2018 USENIX Annual Technical Conference, USENIX ATC 2018, Boston, MA, USA, July 11-13, 2018*, pages 677–690. USENIX Association, 2018.

[32] Jae-Hong Kim, Sangwon Seo, Dawoon Jung, Jin-Soo Kim, and Jaehyuk Huh. Parameter-aware I/O management for solid state disks (ssds). *IEEE Trans. Computers*, 61(5):636–649, 2012.

[33] Jaeho Kim, Donghee Lee, and Sam H. Noh. Towards SLO complying ssds through OPS isolation. In *Proceedings of the 13th USENIX Conference on File and Storage Technologies, FAST 2015, Santa Clara, CA, USA, February 16-19, 2015*, pages 183–189. USENIX Association, 2015.

[34] Jungkil Kim, Sungyong Ahn, Kwanghyun La, and Wooseok Chang. Improving i/o performance of nvme ssd on virtual machines. In *Proceedings of the 31st Annual ACM Symposium on Applied Computing*, pages 1852–1857. ACM, 2016.

[35] Ana Klimovic, Christos Kozyrakis, Eno Thereska, Binu John, and Sanjeev Kumar. Flash storage disaggregation. In *Proceedings of the Eleventh European Conference on Computer Systems*, page 29. ACM, 2016.

[36] Ana Klimovic, Heiner Litz, and Christos Kozyrakis. Reflex: Remote flash $\approx$ local flash. In *Proceedings of the 22nd International Conference on Architectural Support for Programming Languages and Operating Systems, ASPLOS 2017, Xi'an, China, April 8-12, 2017*, pages 345–359. ACM, 2017.

[37] Dongup Kwon, Junehyuk Boo, Dongryeong Kim, and Jangwoo Kim. FVM: FPGA-assisted Virtual Device Emulation for Fast, Scalable, and Flexible Storage Virtualization. In *Proceedings of 14th USENIX Symposium on Operating Systems Design and Implementation (OSDI 20)*, pages 955–971. USENIX Association, 2020.

[38] Jean-Yves Le Boudec and Patrick Thiran. *Network calculus: a theory of deterministic queuing systems for the internet*. Springer, 2001.

[39] Minkyeong Lee, Donghyun Kang, Minho Lee, and Young Ik Eom. Improving read performance by isolating multiple queues in nvme ssds. In *Proceedings of the 11th International Conference on Ubiquitous Information Management and Communication, IMCOM 2017, Beppu, Japan, January 5-7, 2017*, page 36. ACM, 2017.

[40] Chengzhi Li, Almut Burchard, and Jörg Liebeherr. A network calculus with effective bandwidth. *IEEE/ACM Transactions on Networking*, 15(6):1442–1453, 2007.

[41] Huaicheng Li, Mingzhe Hao, Stanko Novakovic, Vaibhav Gogte, Sriram Govindan, Dan R. K. Ports, Irene Zhang, Ricardo Bianchini, Haryadi S. Gunawi, and Anirudh Badam. Leapio: Efficient and portable virtual nvme storage on arm socs. In *Proceedings of the Twenty-Fifth International Conference on Architectural Support for Programming Languages and Operating Systems*, ASPLOS '20, page 591–605, New York, NY, USA, 2020.

[42] Huaicheng Li, Martin L. Putra, Ronald Shi, Xing Lin, Gregory R. Ganger, and Haryadi S. Gunawi. Loda: A host/device co-design for strong predictability contract on modern flash storage. In *Proceedings of the ACM SIGOPS 28th Symposium on Operating Systems Principles*, SOSP '21, page 263–279, New York, NY, USA, 2021. Association for Computing Machinery.

[43] Heiner Litz, Javier Gonzalez, Ana Klimovic, and Christos Kozyrakis. Rail: Predictable, low tail latency for nvme flash. 18(1), jan 2022.

[44] David Lo, Liqun Cheng, Rama Govindaraju, Luiz André Barroso, and Christos Kozyrakis. Towards energy proportionality for large-scale latency-critical workloads. In *Proceedings of the ACM/IEEE 41st International Symposium on Computer Architecture, ISCA 2014, Minneapolis, MN, USA, June 14-18, 2014*, pages 301–312. IEEE Computer Society, 2014.

[45] Jason Mars, Lingjia Tang, Robert Hundt, Kevin Skadron, and Mary Lou Soffa. Bubble-up: increasing utilization in modern warehouse scale computers via sensible colocations. In *Proceedings of the 44rd Annual IEEE/ACM International Symposium on Microarchitecture, MICRO 2011, Porto Alegre, Brazil, December 3-7, 2011*, pages 248–259. ACM, 2011.

[46] Mellanox. Mellanox nvme snap™ | mellanox technologies. https://www.mellanox.com/products/software/nvme-snap, 2020.

[47] Michael Mesnier, Feng Chen, Tian Luo, and Jason B. Akers. Differentiated storage services. In *Proceedings of the Twenty-Third ACM Symposium on Operating Systems Principles*, SOSP '11, page 57–70, New York, NY, USA, 2011. Association for Computing Machinery.

[48] Till Miemietz, Hannes Weisbach, Michael Roitzsch, and Hermann Härtig. K2: work-constraining scheduling of nvme-attached storage. In *Proceedings of the IEEE Real-Time Systems Symposium, RTSS 2019, Hong Kong, SAR, China, December 3-6, 2019*, pages 56–68. IEEE, 2019.

[49] Katherine Missimer and Richard West. Partitioned real-time NAND flash storage. In *Proceedings of the 2018 IEEE Real-Time Systems Symposium, RTSS 2018, Nashville, TN, USA, December 11-14, 2018*, pages 185–195. IEEE Computer Society, 2018.

[50] Mihir Nanavati, Jake Wires, and Andrew Warfield. Decibel: Isolation and sharing in disaggregated rack-scale storage. In *Proceedings of the 14th USENIX Symposium on Networked Systems Design and Implementation, NSDI 2017, Boston, MA, USA, March 27-29, 2017*, pages 17–33. USENIX Association, 2017.

[51] Nvmexpress. Nvm express specification. http://www.nvmexpress.org/specifications/.

[52] Jian Ouyang, Shiding Lin, Song Jiang, Zhenyu Hou, Yong Wang, and Yuanzheng Wang. Sdf: software-defined flash for web-scale internet storage systems. In *Proceedings of ACM SIGARCH Computer Architecture News*, volume 42, pages 471–484. ACM, 2014.

[53] Stan Park and Kai Shen. FIOS: a fair, efficient flash I/O scheduler. In *Proceedings of the 10th USENIX conference on File and Storage Technologies, FAST 2012, San Jose, CA, USA, February 14-17, 2012*, page 13. USENIX Association, 2012.

[54] Bo Peng, Ming Yang, Jianguo Yao, and Haibing Guan. A throughput-oriented nvme storage virtualization with workload-aware management. *IEEE Transactions on Computers*, 70(12):2112–2124, 2020.

[55] Bo Peng, Haozhong Zhang, Jianguo Yao, Yaozu Dong, Yu Xu, and Haibing Guan. Mdev-nvme: A nvme storage virtualization solution with mediated pass-through. In *Proceedings of the 2018 USENIX Annual Technical Conference, USENIX ATC 2018, Boston, MA, USA, July 11-13, 2018*, pages 665–676. USENIX Association, 2018.

[56] Zhiwei Qin, Yi Wang, Duo Liu, and Zili Shao. Real-time flash translation layer for NAND flash memory storage systems. In *Proceedings of the 2012 IEEE 18th Real Time and Embedded Technology and Applications Symposium, Beijing, China, April 16-19, 2012*, pages 35–44. IEEE Computer Society, 2012.

[57] Rusty Russell. virtio: towards a de-facto standard for virtual i/o devices. *ACM SIGOPS Operating Systems Review*, 42(5):95–103, 2008.

[58] Samsung. Enterprise ssd | ssd | samsung semiconductor. https://semiconductor.samsung.com/ssd/enterprise-ssd/pm1733-pm1735/mzplj1t6hbjr-00007/.

[59] Jens B Schmitt, Frank A Zdarsky, and Markus Fidler. Delay bounds under arbitrary multiplexing: When network calculus leaves you in the lurch... In *IEEE INFOCOM 2008-The 27th Conference on Computer Communications*, pages 1669–1677. IEEE, 2008.

[60] Kai Shen and Stan Park. Flashfq: A fair queueing I/O scheduler for flash-based ssds. In *Proceedings of the 2013 USENIX Annual Technical Conference, San Jose, CA, USA, June 26-28, 2013*, pages 67–78. USENIX Association, 2013.

[61] David Shue, Michael J. Freedman, and Anees Shaikh. Performance isolation and fairness for multi-tenant cloud storage. In *Proceedings of the 10th USENIX Symposium on Operating Systems Design and Implementation, OSDI 2012, Hollywood, CA, USA, October 8-10, 2012*, pages 349–362. USENIX Association, 2012.

[62] skkucsl. . https://github.com/skkucsl/d2fq.

[63] Yongseok Son, Hyuck Han, and Heon Young Yeom. Optimizing file systems for fast storage devices. In *Proceedings of the 8th ACM International Systems and Storage Conference*, page 8. ACM, 2015.

[64] Xiang Song, Jian Yang, and Haibo Chen. Architecting flash-based solid-state drive for high-performance I/O virtualization. *IEEE Comput. Archit. Lett.*, 13(2):61–64, 2014.

[65] David Starobinski, Mark Karpovsky, and Lev A Zakrevski. Application of network calculus to general topologies using turn-prohibition. *IEEE/ACM Transactions on Networking*, 11(3):411–421, 2003.

[66] Arash Tavakkol, Mohammad Sadrosadati, Saugata Ghose, Jeremie S. Kim, Yixin Luo, Yaohua Wang, Nika Mansouri-Ghiasi, Lois Orosa, Juan Gómez-Luna, and Onur Mutlu. FLIN: enabling fairness and enhancing performance in modern nvme solid state drives. In *Proceedings of the 45th ACM/IEEE Annual International Symposium on Computer Architecture, ISCA 2018, Los Angeles, CA, USA, June 1-6, 2018*, pages 397–410. IEEE Computer Society, 2018.

[67] Kun Tian, Yaozu Dong, and David Cowperthwaite. A full gpu virtualization solution with mediated pass-through. In *Proceedings of 2014 USENIX Annual Technical Conference (USENIX ATC 14)*, pages 121–132, 2014.

[68] tmiemietz. TUS-OS/k2-scheduler. `https://github.com/TUD-OS/k2-scheduler`.

[69] Kai Wang, Florin Ciucu, Chuang Lin, and Steven H Low. A stochastic power network calculus for integrating renewable energy sources into the power grid. *IEEE Journal on Selected Areas in Communications*, 30(6):1037–1048, 2012.

[70] Alex Williamson. Vfio: A user's perspective. `https://www.linux-kvm.org/images/b/b4/2012-forum-VFIO.pdf`.

[71] Jiwon Woo, Minwoo Ahn, Gyusun Lee, and Jinkyu Jeong. D2FQ: Device-Direct fair queueing for NVMe SSDs. In *Proceedings of 19th USENIX Conference on File and Storage Technologies (FAST 21)*, pages 403–415. USENIX Association, 2021.

[72] Jie Zhang, Miryeong Kwon, Donghyun Gouk, Sungjoon Koh, Changlim Lee, Mohammad Alian, Myoungjun Chun, Mahmut Taylan Kandemir, Nam Sung Kim, Jihong Kim, and Myoungsoo Jung. Flashshare: Punching through server storage stack from kernel to firmware for ultra-low latency ssds. In *Proceedings of the 13th USENIX Symposium on Operating Systems Design and Implementation, OSDI 2018, Carlsbad, CA, USA, October 8-10, 2018*, pages 477–492. USENIX Association, 2018.

[73] Fam Zheng. Userspace nvme driver in qemu. `https://events.static.linuxfound.org/sites/events/files/slides/Userspace%20NVMe%20driver%20in%20QEMU%20-%20Fam%20Zheng_0.pdf`.

[74] Timothy Zhu, Alexey Tumanov, Michael A Kozuch, Mor Harchol-Balter, and Gregory R Ganger. Prioritymeister: Tail latency qos for shared networked storage. In *Proceedings of the ACM Symposium on Cloud Computing*, pages 1–14, 2014.