Future Generation Computer Systems 173 (2025) 107902

Contents lists available at ScienceDirect

Future Generation Computer Systems

journal homepage: www.elsevier.com/locate/fgcs

State-driven fairness control for efficient I/O queue scheduling in NVMe virtualization

Zhaoyang Huang [a], Yifu Zhu [a], Xin Kuang [a], Yanjie Tan [a], Huailiang Tan [a,*], Keqin Li [a,b]

[a] College of Computer Science and Electronic Engineering, Hunan University, Changsha, 410082, Hunan, China
[b] Department of Computer Science, State University of New York, New Paltz, 12561, NY, USA

ARTICLE INFO

Keywords:
Fairness
I/O scheduling
Multi-tenant cloud
NVMe SSD
Storage virtualization

ABSTRACT

As data centers and cloud environments expand, enhancing fairness in I/O queue resource scheduling has become increasingly urgent in the field of Non-Volatile Memory Express (NVMe) storage virtualization. Existing methods usually focus on metrics such as Input/Output Operations Per Second (IOPS) enhancement or latency reduction, overlooking fairness issues among virtual machines (VMs) which may lead to significant resource contention and performance degradation. In this paper, we propose FairNVMe, a novel NVMe virtualization solution that enables fair I/O queue scheduling among multiple tenants through effective fairness control. FairNVMe introduces a state-driven fairness controller that assigns a private state for each tenant and triggers adaptive resource adjustments when unfair tenant states are detected. Specifically, FairNVMe employs time budget-based I/O queue scheduling with dynamic budget compensation, reallocating time budgets based on actual resource consumption and requirements of each tenant. Experimental results demonstrate that FairNVMe alleviates mutual competition among multiple tenants and outperforms existing solutions in terms of both system performance and fairness, reaching up to 94.5%, 61.2%, and 79.5% tail latency optimization, enhancing the fairness by up to 51.4%, 15.6%, and 73.2%, and mitigates maximum slowdowns by up to 92.1%, 59.2%, and 58.2% compared with Virtio, SPDK, and LPNS, respectively.

1. Introduction

The rapid advancement of new-generation information technologies such as 5G, cloud computing, and artificial intelligence has rendered data centers a crucial component for modern information infrastructure [1,2]. Simultaneously, the volume of data transmitted and processed within cloud environments is experiencing explosive growth, presenting challenges in data management and processing efficiency [3]. To mitigate these challenges and achieve faster data transmission speeds with lower latency, NVMe devices are widely deployed in data centers and cloud platforms [4–6].

NVMe storage virtualization has emerged as a prominent area of research due to its potential to enhance the effectiveness and scalability of storage devices while reducing the operational costs of data centers [7,8]. Traditional storage virtualization technologies, such as VirtIO [9], H-NVMe [10], FamZ userspace NVMe driver [11], and Storage Performance Development Kit (SPDK) [12] employ universal Linux I/O virtualization frameworks. These methods involve complex software stacks and may not be fully optimized to exploit the high-speed capabilities of NVMe devices, resulting in potential processing inefficiencies. MDev-NVMe [13] introduces a full NVMe virtualization framework with mediated pass-through, where each guest runs a native NVMe driver and facilitates device sharing. By passing through performance-critical resources, MDev-NVMe increases the utilization and scalability of storage devices. For enhanced workload-aware management and latency-predictable I/O control, Peng et al. further propose FinNVMe [14] and LPNS [15], which provide improved I/O throughput and operational efficiency within virtualized environments. However, they fail to ensure fair I/O queue resource allocation among tenants. Hardware-assisted approaches, such as Single Root I/O Virtualization (SR-IOV) [16], LeapIO [17], and FVM [18], enable the partitioning of physical I/O devices into virtual functions, facilitating direct access by multiple VMs. Nevertheless, these solutions rely on dedicated hardware support and exhibit limited flexibility, constraining their deployment by cloud vendors across diverse hardware platforms.

With the sustained increase in cloud users, improving the fairness of physical resource allocation and maintaining the stability of tenant request processing has become an urgent challenge [19–22]. Co-located

* Corresponding author.
E-mail address: tanhuailiang@hnu.edu.cn (H. Tan).

https://doi.org/10.1016/j.future.2025.107902
Received 22 January 2025; Received in revised form 7 April 2025; Accepted 4 May 2025
Available online 20 May 2025

Table 1
Related work of storage virtualization solutions.

Technology	Methods	High Performance	Flexible I/O Scheduling	Fairness Control	Description and Limitation
PT	VFIO [29]	✓	×	×	Pass-through technology assigns the entire NVMe device to a single VM, failing to achieve device sharing.
Para	Virtio [9]	×	×	×	Virtio lacks specific optimization for NVMe devices. SPDK relies on Hugepage memory, imposing additional pressure.
	SPDK [12]	✓	×	×	
SR-IOV	FVM [18]	✓	×	×	Hardware-assisted virtualization relies on dedicated hardware support and exhibit limited flexibility.
	LeapIO [17]	×	×	×	
MPT	Mdev-NVMe [13]	✓	×	×	Existing mediated pass-through NVMe virtualization focuses on I/O performance enhancement and fail to address the critical issue of fair I/O queue resource allocation.
	FinNVMe [14]	✓	✓	×	
	LPNS [15]	✓	✓	×	
	FairNVMe (Ours)	✓	✓	✓	The first NVMe virtualization with effective fairness control.

* PT: Pass-through. MPT: Mediated Pass-through.

tenants share underlying physical NVMe devices, leading to inevitable competition and potential interference. To address these issues, the Linux kernel introduces several I/O fair scheduling algorithms, such as CFQ [23], BFQ [24], and mq-deadline [25]. While effective in traditional systems, these algorithms fail to fully perceive distinct I/O load characteristics of individual virtual machines, making it hard to achieve optimal scheduling in dynamic virtualization environments. Recent research has introduced device-level fair I/O scheduling strategies tailored for NVMe SSDs [26–28]. FLIN [26] performs an in-depth analysis of interference in multi-queue SSDs and proposes a flash-level interference-aware scheduler that aims to balance slowdowns while adhering to application-level priorities assigned by the host system. Fuzzy [27] focuses on the interference at the data cache level of the SSD and presents a fuzzy logic-based fairness controller. Fair-ZNS [28] targets the specialized NVMe Zoned Namespace (ZNS) storage interface and introduces a self-balance I/O scheduling mechanism dedicated to ZNS SSDs. Unfortunately, these fairness control strategies often require modifications to the underlying SSD controller, which increases the algorithm complexity and implementation cost, restricting their deployment in practical data centers.

Existing NVMe storage virtualization techniques fail to address the critical issue of fair I/O queue resource allocation in multi-tenant cloud storage systems with constrained resources. It may result in severe imbalances in system efficiency and degrade the quality of service (QoS) delivered to affected tenants, failing to meet their performance expectations and negatively impacting user satisfaction [30–32]. In response to these challenges, we introduce FairNVMe, a novel NVMe virtualization solution that incorporates effective fairness control mechanisms to enable flexible I/O queue scheduling and mitigate resource contention among multiple tenants. Firstly, FairNVMe classifies tenant states into four distinct levels based on predefined latency thresholds and measured I/O latency, facilitating real-time assessment of tenant load and system fairness conditions. Secondly, FairNVMe employs a state-driven fairness control strategy that continuously monitors the runtime state of each tenant and generates control signals to initiate adaptive resource adjustments when state transitions are detected. Finally, the system integrates a time budget-based I/O queue scheduling approach with dynamic budget compensation. This mechanism allocates specific time budgets to tenants and dynamically adjusts them based on request cost estimation, thereby offering a comprehensive understanding of each tenant's actual resource consumption and requirements.

The main contributions of our work are as follows.

- We present FairNVMe, a novel OS-level mediated pass-through NVMe virtualization solution with effective fairness control. FairNVMe guarantees fair I/O queue scheduling among multiple tenants, addressing the critical issue of resource contention in multi-tenant clouds.
- We introduce a state-driven fairness controller that continuously upholds an individual state for each tenant. The controller triggers an unfairness signal whenever it identifies unbalanced or saturated states among tenants. To ensure balanced resource distribution, we estimate the time cost of each read/write operation and integrate time budget-based I/O scheduling with dynamic budget compensation.
- We conduct an extensive series of experiments to compare the proposed FairNVMe with three mainstream NVMe virtualization frameworks, including para-virtualization Virtio, SPDK, and LPNS with mediated pass-through. Experimental results demonstrate that FairNVMe maintains the high performance of NVMe devices while effectively managing fairness among multiple tenants, outperforming existing solutions in terms of both system performance and fairness.

The remainder of this paper is organized as follows. Section 2 provides a comprehensive overview of the background and motivation. In Section 3, we illustrate the design and implementation of FairNVMe in detail. Section 4 presents the discussion. Experimental results compared the proposed FairNVMe architecture with Virtio, SPDK, and LPNS are demonstrated in Section 5. Section 6 introduces related works. Finally, the conclusion is drawn in Section 7.

2. Background and motivation

2.1. NVMe protocol

NVMe is an advanced storage access and transport protocol specifically designed for SSD devices [4]. NVMe operations can be classified into two primary categories: Admin Commands and I/O Commands. Admin Commands undertake administrative tasks such as I/O queue management, device parameter configuration, and function management. On the other hand, I/O Commands are responsible for data transmission between the host system and SSD.

The NVMe architecture is built around three principal components: Submission Queue (SQ), Completion Queue (CQ), and Doorbell Register

(DB). The SQ serves as the host system's interface for dispatching commands to the SSD, encompassing both Admin and I/O Commands. CQ receives updates and completion notifications from the SSD, providing essential feedback to the host system. The DB plays a crucial role in signaling between the host and the SSD device. When the host system enqueued Admin and I/O Commands into the SQ, it rings the doorbell to alert the SSD hardware controller of pending tasks. The controller periodically checks for modifications in the doorbell register and retrieves commands from the SQ for execution. Upon completing command execution, the controller writes relevant information to the CQ and triggers a Message Signaled Interrupt (MSI/MSI-X) in the DB to notify the host system of task completion. After receiving the interrupt, the host system reads the information from the CQ, updates the command status, and prepares for subsequent command processing.

2.2. Fairness definition

To quantify the fairness of NVMe virtualization, we follow the definition in prior research [26,28,33]. Fairness is defined as the ratio between the minimum slowdown value and the maximum slowdown value, formulated as follows:

$$Slowdown_i = \frac{RL_i^{Shared}}{RL_i^{Alone}}, \tag{1}$$

$$Fairness = \frac{\min_i\{Slow_i\}}{\max_i\{Slow_i\}}. \tag{2}$$

When multiple I/O flows run simultaneously, they can negatively interfere with each other, leading to performance degradation. The slowdown metric $Slowdown_i$ measures the performance loss experienced by each I/O flow due to resource contention. Specifically, RL_i^{Shared} refers to the request latency of the ith I/O flow when it operates under shared conditions, where multiple I/O flows run concurrently. RL_i^{Alone} denotes the request latency when the flow is running alone. The fairness metric $Fairness$, derived from these slowdown values, provides a comprehensive measure of the equitability of resource allocation among multiple I/O flows. $Fairness$ ranges from 0 to 1, where a higher value indicates a more balanced system. The fairness value approaching 1 signifies an ideal system condition where all I/O flows experience similar slowdown levels due to resource contention, implying that the system facilitates fair resource distribution.

2.3. Motivation

Table 1 categorizes existing NVMe virtualization mechanisms [9,12–15,17,18,29] into four categories, including pass-through, para-virtualization, hardware-assisted virtualization, and mediated pass-through. Although existing NVMe virtualization methods have achieved reasonable efficiency, they fail to address the following critical challenges inherent in cloud environments.

(1) Performance degradation. When multiple co-located tenants simultaneously access shared underlying NVMe devices, the contention for I/O resources inevitably leads to increased latency and reduced throughput. This slowdown has a direct and adverse impact on the QoS experienced by each tenant, as tenants may face delays in data processing, which can be particularly detrimental to time-sensitive applications. For instance, in financial services, where real-time transaction processing is essential, any lag can lead to substantial operational challenges and financial losses. To quantify the slowdown caused by mutual interference among tenants, we evaluate the IOPS, average latency, and corresponding slowdown metrics for each virtual machine running intensive 4K random read and write operations. The detailed experimental setup is described in Section 5.1. As depicted in Fig. 1, the performance of each VM utilizing Virtio and LPNS exhibits varying levels of degradation, highlighting resource contention and mutual interference in multi-tenant environments. Although SPDK preserves a relatively consistent slowdown among VMs, it still suffers from severe degradation in system efficiency under high contention scenarios.

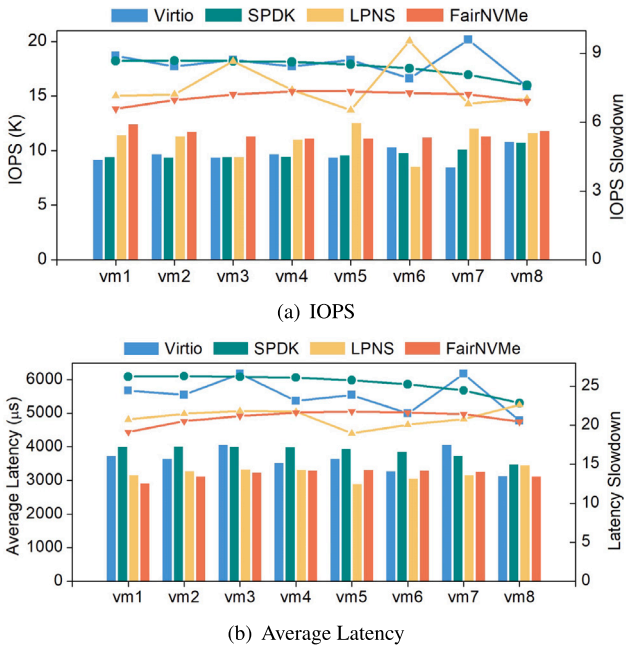


Fig. 1. System performance (histogram) and corresponding slowdown (line graph) caused by mutual resource contention among multiple VMs.

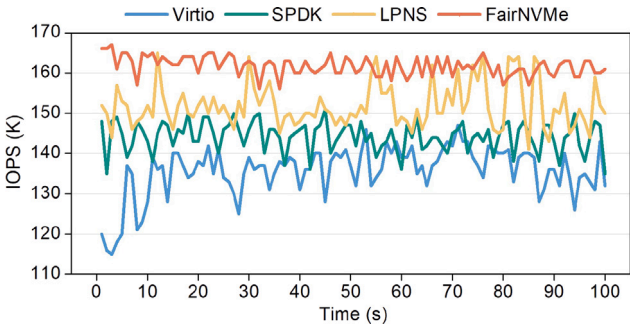


Fig. 2. The stability of I/O throughput under real-world application scenarios.

(2) Instability. System instability and fluctuations introduced by resource contention can complicate resource management and lead to inconsistent application behaviors. The performance of individual tenants can vary unpredictably, which poses challenges in preserving consistent service levels. To assess system stability under real-world scenarios, we assign two VMs to execute application workloads webserver and fileserver generated by Filebench [34]. The other two VMs run intensive FIO workloads, characterized by random read and random write operations, configured with numjobs = 4 and iodepth = 32. As illustrated in Fig. 2, we observe the system IOPS over a 100-second interval. The experimental results reveal that the throughput of Virtio, SPDK, and LPNS fluctuates significantly over time due to resource contention among co-located tenants. As evaluated in Section 5.3, we also quantify the stability of tenant latency using the standard deviation metric and observe that resource contention also contributes to substantial latency variability. Such instability might make it hard to ensure the reliability and predictability of cloud services [35].

(3) Unfairness in resource allocation. Unfair resource allocation can lead to scenarios where some tenants experience severe degradation while others benefit disproportionately from the shared resources. For example, as depicted in Fig. 1(a), the IOPS throughput of VM_6 using

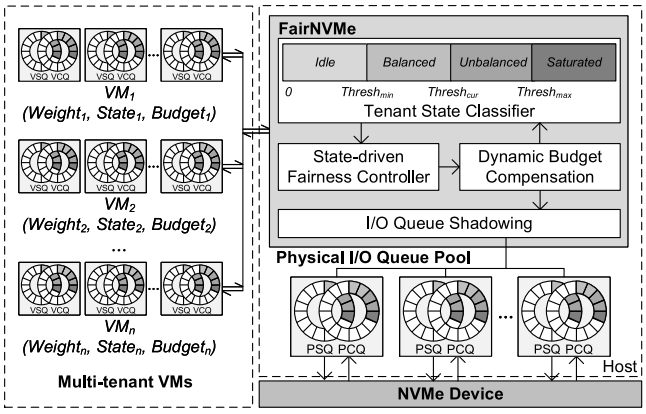


Fig. 3. System architecture of FairNVMe.

LPNS experiences considerably higher slowdowns compared to VM_5. This imbalance considerably affects the overall efficiency and fairness of cloud environments. Maintaining fairness in resource distribution is essential for enhancing tenant satisfaction and optimizing the utilization of cloud resources. To address the above critical challenges and enable fair NVMe virtualization, we introduce a state-driven fairness controller and integrate time budget-based I/O scheduling with dynamic budget compensation.

3. Design and implementation

In this section, we introduce FairNVMe, a novel NVMe virtualization framework with effective fairness control to mitigate mutual interference among multiple tenants and enable fair I/O queue scheduling.

3.1. Architecture overview

FairNVMe is designed based on mediated pass-through technique [36] and implemented as a kernel module on the Linux host system (kernel version 5.5.0). FairNVMe offers full virtualization support for NVMe devices without necessitating any modifications to the guest OS kernel. The system architecture is depicted in Fig. 3. For simplicity in model design, we assume that each tenant contains only a single VM, with fairness achieved at the VM level. Nevertheless, the proposed model can be easily extended to other scenarios where tenants hold multiple VMs. FairNVMe inherits the physical I/O queue pool and I/O queue shadowing design principles from previous mediated pass-through NVMe virtualization solutions [13,14]. In FairNVMe, each tenant is allocated virtual I/O queues, which comprise virtual submission queues (VSQs) and virtual completion queues (VCQs). Commands from each tenant are initially delivered to the VSQs and then passed through to the shadowing physical queue via Direct Memory Access (DMA) and the translation of Guest Physical Address (GPA) to Host Physical Address (HPA). To achieve fairness in I/O queue management, FairNVMe presents several key components: a tenant state classifier, a state-driven fairness controller, and a dynamic budget compensation module.

The tenant state classifier categorizes tenant states into four distinct levels based on predefined thresholds: idle, balanced, unbalanced, and saturated. Each category reflects the current load and performance status of tenants' I/O operations. To flexibly adapt to tenant state transitions, FairNVMe employs dedicated polling threads to handle I/O commands while continuously monitoring the status of each tenant. Upon detecting unfairness or overload conditions, the fairness controller adjusts the latency thresholds and associated weights while activating the budget compensation process to maintain system fairness and efficiency. The budget compensation module estimates the request cost associated with read/write operations, enabling resource allocation strategies to be adjusted according to the actual resource consumption and demands. Further details on the implementation and functionality of these components will be presented in the following subsection.

3.2. Tenant state classifier

Accurate assessments of each tenant's fairness state are crucial for implementing effective fairness control. Motivated by research presented in [37,38], which define fairness as the variance between the expected and actual weight of each tenant. We utilize the difference between measured latency and predefined target levels as an indicator of potential unfairness. However, determining suitable latency thresholds poses challenges. The performance characteristics of each virtual machine are unpredictable and influenced by a multitude of factors, including request size (average and distribution), operation mix (the percentage of read versus write operations), and workload access patterns (sequential or random). These dynamic variations make it hard for a fixed latency threshold to be universally applicable across diverse scenarios. In response, we introduce an adaptive latency threshold adjustment mechanism that dynamically updates the threshold value, denoted as $Thresh_{cur}$, by leveraging the exponentially weighted moving averages (EWMA) [39].

Tenant states are further categorized into four distinct levels based on measured latency $Latency$ in relation to the minimum threshold $Thresh_{min}$, the adaptive current threshold $Thresh_{cur}$, and the maximum threshold $Thresh_{max}$. (1) Idle ($Latency < Thresh_{min}$): allocated resources are underutilized. (2) Balanced ($Thresh_{min} \leq Latency < Thresh_{cur}$): the ideal state for achieving optimal system efficiency and effective resource management. (3) Unbalanced ($Thresh_{cur} \leq Latency < Thresh_{max}$): suggesting that the current resource distribution may be suboptimal, potentially leading to system unfairness and inefficiencies. (4) Saturated ($Latency \geq Thresh_{max}$): the demand for system resources surpasses the available capacity, resulting in significant performance delays. Typically observed in scenarios with excessive task volumes or insufficient resource allocation.

As the lower bound of *Balanced* state, $Thresh_{min}$ must exceed the maximum latency observed when the I/O flow operates in isolation, i.e., $max(RL_{Alone})$. We configure $Thresh_{min}$ to 600 µs to ensure the system remains within the *Balanced* state under light load conditions. The upper threshold $Thresh_{max}$ is critical in enabling efficient resource utilization under heavy loads while avoiding frequent transitions into the *Saturated* state, which could compromise the system stability. To strike the optimal balance, we set $Thresh_{max}$ to 3500 µs, which is slightly higher than the latency observed as the device approaches saturation. Moreover, to effectively accommodate dynamic workloads and diverse SSD hardware characteristics, we adopt a feedback loop-based threshold adjustment strategy. Inspired by the adaptive threshold design in GCC [40], we continuously monitor latency trends and dynamically update the minimum and maximum threshold using the following equation:

$$Thresh(t_i) = Thresh(t_{i-1}) + k \cdot (Latency(t_i) - Thresh(t_{i-1})), \tag{3}$$

$$k = \begin{cases} k_u, & Latency(t_i) > Thresh(t_{i-1}), \\ k_d, & otherwise. \end{cases} \tag{4}$$

The adaptation parameters k_u and k_d control the rates of threshold increase or decrease. Similar to GCC [40], we employ an asymmetric adaptation ratio ($k_u = 0.01 \gg k_d = 0.001$) to ensure rapid response to latency increases while maintaining smooth recovery during stable periods. Our parameter sensitivity analysis also confirms that this configuration optimally balances system responsiveness and stability across diverse SSD workloads. Notably, the computational overhead of this adaptive mechanism is negligible, as it only requires simple arithmetic operations without introducing additional I/O operations or costly computations.

Algorithm 1 Fair I/O Queue Scheduling through State-Driven Fairness Control

Input: Number of tenants N, predefined weight $weight[N]$, maximum and minimum threshold $thresh_{max}$, $thresh_{min}$
Output: Budget allocation results for each tenant $budget[N]$

```
1: for i = 1 to N
2:   while req[i] > 0
3:     if budget[i] > 0 then
4:       latency[i] = request_processing(req);
5:       budget[i] -= cost[i];
6:       if state[i] == Idle && weight[i] > β then
7:         weight[i] -= β;
8:         thresh[i] = ewma(latency, thresh);
9:       else if state[i] == Balanced then
10:        /* Keep the current weight and time budget */
11:        thresh[i] = ewma(latency, thresh);
12:      else if state[i] == Unbalanced then
13:        weight[i] += β;
14:        thresh[i] = (thresh[i] + thresh_max[i]) / 2;
15:        cost = cost_estimation(req);
16:        budget = budget_compensate(weight, cost);
17:        return budget;
18:      else
19:        weight[i] += 2β;
20:        thresh[i] = thresh_max[i];
21:        cost = cost_estimation(req);
22:        budget = budget_compensate(weight, cost);
23:        return budget;
24:      end if
25:    else /* Tenant exhausts allocated time budgets */
26:      Break; /* Excluded from the scheduling unit */
27:      if i==N /* All tenants run out of budgets */
28:        budget = budget_reallocate(weight, cost);
29:        return budget;
30:      end if
31:    end if
32:  end while
33: end for
34: End
```

3.3. State-driven fairness controller

For flexible I/O queue scheduling, FairNVMe integrates a time budget-based I/O queue scheduling mechanism with a state-driven fairness controller. Specifically, FairNVMe continuously monitors the tenant status to identify varying levels of unfairness, guiding the system to apply either aggressive or gradual time budget adjustments based on the severity of the imbalance.

The procedure for the fair I/O queue scheduling is outlined in Algorithm 1, which exhibits a time complexity of $O(N \cdot max(req_i))$. This complexity arises from the nested loop structure: the outer loop iterates over N tenants, while the inner loop processes up to $max(req_i)$ requests for each tenant. Since operations within the inner loop have a constant time complexity, the total number of operations in the worst case is proportional to $N \cdot \max(req_i)$. Initially, each tenant is assigned a default weight of 1 upon creation, which determines its proportion of resource allocation and is subsequently dynamically adjusted based on the tenant's status. The system then fairly allocates time budgets among tenants according to their associated weights. As I/O requests are processed, the time budget is incrementally reduced based on the estimated costs (Steps 1 to 5). During execution, the system continuously monitors the runtime state of each tenant according to their measured I/O latency and corresponding thresholds. Upon detecting state transitions, the algorithm dynamically modifies the relevant weights and thresholds. Additionally, it triggers cost estimation and budget compensation processes (as described in Section 3.4) when necessary to maintain system fairness and efficiency (Steps 6 to 24). Once a tenant's allocated time budget is exhausted, their request queue is temporarily excluded from the scheduling unit until the budget is replenished in the next scheduling cycle, preventing any single tenants from monopolizing system resources and ensuring that all tenants have equitable access over time (Steps 25 to 26). In scenarios where all tenants exhaust their time budgets and there are still pending requests, the scheduler resets the time budgets, allowing the system to continue processing I/O operations without introducing significant delays (Steps 27 to 34). Detailed explanations of the control actions associated with each state are provided below.

(1) Releasing unused resources during the idle state. Idle state indicates the potential for redistributing unused capacity to other tenants experiencing higher demand, particularly those in unbalanced or saturated states for overall performance optimization. To facilitate this, we introduce a new parameter β for tenant resource allocation weight adjustments in response to state transitions. Specifically, the fairness controller reduces the weight associated with idle tenants by β and updates the current threshold using the EWMA method (Steps 6 to 8).

(2) Preserving optimal efficiency in the balanced state. The fairness controller preserves the current resource allocation strategy as no immediate adjustments are necessary (Steps 9 to 11).

(3) Gradually mitigating unfairness in the unbalanced state. This state serves as an early warning that system fairness is facing degradation. In response, the algorithm implements proactive actions to correct these imbalances before they escalate into more severe issues (i.e., the saturated state). Initially, the system gradually increases the weight associated with unbalanced tenants by β to enhance its I/O queue resource allocation ratio (Steps 12 to 13). To reduce the frequency of transitions into the unbalanced state, the algorithm incrementally adjusts the $Thresh_{cur}$ to the midpoint between the current threshold and the maximum threshold (Step 14). Additionally, it triggers cost estimation and budget compensation processes to maintain system fairness (Steps 15 to 17).

(4) Aggressively addressing system stress in the saturated state. The tenant load is either approaching or exceeding the system's processing capacity, necessitating immediate intervention to prevent further system degradation and ensure stability. Specifically, we aggressively increase the weight associated with saturated tenants by 2β, thereby prioritizing their resource allocation (Steps 18 to 19). Moreover, the controller activates the dynamic budget compensation process and updates the current threshold to align with the maximum threshold value, preventing further performance degradation and enabling the system to quickly adapt to extreme load conditions (Steps 20 to 24).

To ensure accurate control, we perform a sensitivity analysis on the parameter β and observe that smaller values of β potentially delay the resolution of unfairness among tenants. Conversely, a larger β enables more aggressive adjustments in resource allocation but heightens the risk of overcompensation, which could exacerbate latency issues. The system latency first declines and then rises as the β value varies, reaching optimal performance at $\beta = 0.5$, which enables effective response to tenant state changes without excessive compensation. By applying corresponding actions for these four states, the fairness controller can rapidly detect and respond to each tenant's resource adjustment requirements, thereby enhancing system stability and fairness across all tenants.

3.4. Dynamic budget compensation

To enable the adaptive adjustment of resource allocation strategy according to the actual resource consumption and demands of each tenant, the budget compensation module estimates the request costs associated with each I/O operation. Given that read and write requests exhibit distinct characteristics and consequently impose varying levels

Table 2
Overhead in average latency (μs)

Test Case	Native	Virtio	SPDK	LPNS	Ours
rand-read-n1d1	99.94	209.32	103.27	101.39	99.5
rand-write-n1d1	14.75	70.87	18.48	15.47	15.73
rand-read-n4d4	147.03	172.54	130.47	121.91	123.47
rand-write-n4d4	53.66	150.29	53.81	54.36	53.73

of resource consumption. Write requests generally exhibit higher latency and demand more extensive processing resources compared to read requests. FairNVMe estimates the costs of read and write operations independently. The estimator gathers detailed I/O request data for each $Tenant_i$, including the number of requests Req_i, the access modes (read or write), and the corresponding time cost, denoted as $Time_i^r$ for read operations and $Time_i^w$ for write operations. Upon detecting signals of unfairness or overload, FairNVMe calculates the associated request costs and initiates the budget compensation process using the following predefined models:

$$Budget_i = \frac{Weight_i \times (Cost_i^r + Cost_i^w)}{\sum_{j=1}^{N} Weight_j \times (Cost_j^r + Cost_j^w)} \times B_{total}, \tag{5}$$

$$Cost_i^r = Req_i \times (1 - Ratio_i^w) \times Time_i^r, \tag{6}$$

$$Cost_i^w = Req_i \times Ratio_i^w \times Time_i^w. \tag{7}$$

The budget allocation for each $Tenant_i$ denoted as $Budget_i$ is determined in proportion to its assigned weight $Weight_i$. The total available time budget is represented as B_{total}. Consumed budgets for completing read and write requests are denoted as $Cost_i^r$ and $Cost_i^w$, respectively. The parameter $Ratio_i^w$ reflects the ratio of write requests relative to the total number of I/O requests Req_i. Through dynamic budget compensation, FairNVMe ensures that tenants with higher weights or greater resource demands receive an appropriate share of the total available budget.

4. Discussion

4.1. Overhead of FairNVMe

FairNVMe implements a range of advanced mechanisms to ensure fair I/O queue scheduling in multi-tenant environments. Nevertheless, their implementation inherently introduces certain performance and resource overheads. Firstly, FairNVMe necessitates continuous monitoring of system status and dynamic adjustment of time budgets allocated to each tenant. This process involves frequent calculations and updates to the time budgets based on runtime statistics, which inevitably contribute to additional latency within the system. Table 2 presents a comparative analysis of the native behavior of physical devices against the average latency observed by Virtio, SPDK, LPNS, and FairNVMe. Experimental results demonstrate that FairNVMe can achieve near-native efficiency and the additional overhead induced by FairNVMe is negligible. Notably, FairNVMe even surpasses native results in 4K random read I/O benchmarks configured with numjobs = 4 and iodepth = 4, owing to its ability to fully exploit multi-queue features of NVMe devices through polling mode.

Secondly, FairNVMe continuously monitors a private state for each tenant and employs dedicated polling threads to facilitate detailed statistics gathering, which introduces additional overhead in terms of CPU cycles and memory utilization. To thoroughly evaluate the resource overhead of FairNVMe, we increase the number of tenants from 1 to 8 and assess the corresponding changes in resource utilization. Our measurements reveal that the additional resource occupancy introduced by FairNVMe remains minimal and within acceptable bounds, with an observed increase of less than 1.8 percent.

4.2. Comparison with device-level fair scheduler

Since the firmware of commercial products is typically closed and cannot be modified, existing device-level I/O schedulers are usually implemented using SSD simulators [41,42]. While these simulators effectively model the internal logic and I/O performance of SSDs, they do not expose block device interfaces to the host or virtualization environments. This limitation makes it unsuitable for direct comparison with FairNVMe, as our work focuses on NVMe virtualization solutions aimed at mitigating performance interference and ensuring fairness among multiple tenants. Therefore, in this subsection, we perform a theoretical analysis to examine the advantages and limitations of FairNVMe in comparison with FLIN [26], a representative device-level scheduling framework.

FLIN is a device-level interference-aware scheduler implemented within the SSD controller firmware using the MQSim simulator [43]. It protects against the four primary sources of interference among I/O flows (i.e., I/O intensity, access pattern, read/write ratio, and garbage collection) to balance slowdowns and ensure fairness across diverse applications. Although device-level solutions like FLIN are expected to achieve better fairness through fine-grained flow control, they are constrained by the following limitations: (1) Implementation complexity: Designing and deploying them requires modifications to the underlying SSD controller firmware, which increases the algorithm complexity and implementation costs [15]. (2) Limited Portability: They are often tightly coupled with specific hardware, making them less portable or reusable across diverse platforms [44]. (3) Scalability issues: In distributed or cloud environments involving multiple devices, device-level schedulers might lack a system-wide perspective, potentially leading to suboptimal global fairness [45]. In contrast, while FairNVMe cannot directly perceive and address some underlying sources of interference, such as garbage collection, it offers notable advantages as an OS-level software strategy. As a kernel module, FairNVMe offers enhanced flexibility for modifications and updates. Furthermore, it can enforce fairness across the entire system, which is crucial in multi-tenant cloud environments.

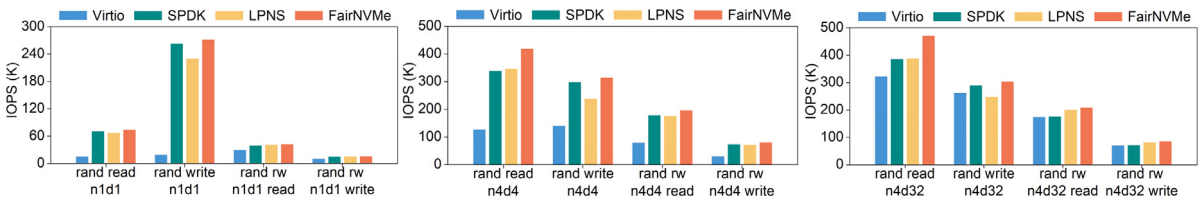
5. Evaluation

In this section, we conduct a comprehensive evaluation of our proposed FairNVMe alongside three mainstream NVMe virtualization mechanisms, including the para-virtualization abstraction Virtio [9], the SPDK *vhost-scsi* [12], and a mediated pass-through solution LPNS [15]. Since LPNS has already demonstrated superior effectiveness over SR-IOV [16], we only need to compare FairNVMe with LPNS in the experiments. Our assessment covers a range of critical metrics, including overall system performance (IOPS, average latency, and tail latency distribution), and fairness (fairness index, maximum slowdown, and system stability). To assess the scalability of FairNVMe across diverse load situations, we augment the workload intensity by increasing the number of jobs and I/O queue depths across various scenarios, including random read, random write, and mixed random read&write conditions.

5.1. Experimental setup

(1) Experimental Platform

The host server is equipped with an Intel Xeon(R) Platinum 8171M 52-core processor running at 2.6GHz, 128GB DRAM. To simulate CPU resource contention scenarios, the number of available CPU cores are constrained to 24. We build the experimental setup using the Intel DC P3520 1.2T NVMe SSD, which is a 3D NAND-based SSD designed specifically for data centers. Different from conventional flash-based SSDs such as the P3700 series, which focuses on high-performance storage, the 3D NAND-based P3520 can realize higher density at lower



(a) I/O scarce (b) I/O moderate (c) I/O intensive

Fig. 4. Throughput performance of IOPS under 12 test cases across three distinct I/O patterns.

Table 3
Workload configuration.

Test Case		Description
I/O scarce	rand-read-n1d1	4K random read, numjobs = 1, iodepth = 1
	rand-write-n1d1	4K random write, numjobs = 1, iodepth = 1
	rand-rw n1d1-read	4K mixed random read&write, numjobs = 1, iodepth = 1, read 70%
	rand-rw n1d1-write	4K mixed random read&write, numjobs = 1, iodepth = 1, write 30%
I/O moderate	rand-read-n4d4	4K random read, numjobs = 4, iodepth = 4
	rand-write-n4d4	4K random write, numjobs = 4, iodepth = 4
	rand-rw n4d4-read	4K mixed random read&write, numjobs = 4, iodepth = 4, read 70%
	rand-rw n4d4-write	4K mixed random read&write, numjobs = 4, iodepth = 4, write 30%
I/O intensive	rand-read-n4d32	4K random read, numjobs = 4, iodepth = 32
	rand-write-n4d32	4K random write, numjobs = 4, iodepth = 32
	rand-rw n4d32-read	4K mixed random read&write, numjobs = 4, iodepth = 32, read 70%
	rand-rw n4d32-write	4K mixed random read&write, numjobs = 4, iodepth = 32, write 30%

cost and power consumption [46]. To comprehensively assess the system fairness, we also deploy FairNVMe on the Samsung PM1735 [47], which is an enterprise-grade SSD capable of supporting up to 1000K IOPS in 4K random read and 200K IOPS in 4K random write.

We install the Ubuntu 18.04 operating system for the host server with Linux kernel 5.5.0, guests are instantiated with identical OS image versions based on KVM/QEMU hypervisor infrastructure. Virt-manager is used for VM creation and management, ensuring streamlined orchestration of virtualized environments. Each virtual machine is endowed with 4 CPU cores and configured to accommodate 4 virtual queues. During experiments, we allocate fixed 64GB NVMe SSD storage to each VM.

(2) Workload Configuration

Flexible I/O Tester (FIO) [48] and Yahoo! Cloud Serving Benchmark (YCSB) [49] are utilized as benchmarking tools to assess system performance comprehensively. To analyze system behavior from multiple dimensions, we adjust FIO configuration parameters and design 12 test cases across three I/O patterns, including I/O scarce, I/O moderate, and I/O intensive. Configuration details are summarized in Table 3. We augment the workload intensity by manipulating the value of *numjobs* (i.e., the amount of concurrent threads) and *iodepth* (i.e., the number of requests queued per thread). Aligned with real-world data center benchmarks derived from CAMELab Flash-based Block Traces (CAMETBT) [50], we set the block size across all test cases as $4K$. The *ioengine* parameter is designated as *libaio*, enabling enhanced asynchronous I/O operations. In pursuit of realistic test outcomes, we employ *Direct* mode to bypass the I/O buffer. Moreover, YCSB is employed to simulate key–value store workloads on RocksDB databases, providing insights into the system's efficiency under realistic application scenarios.

5.2. Overall performance

(1) Throughput

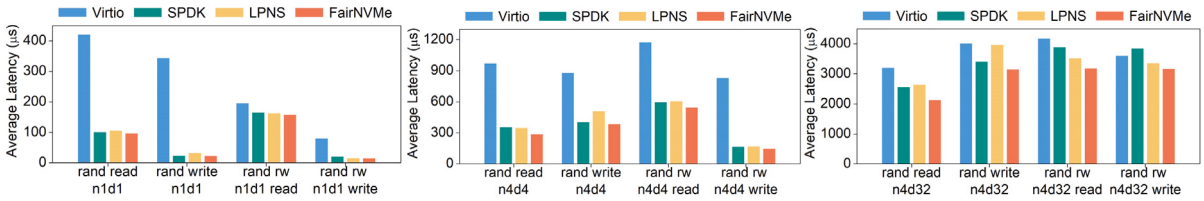
Initially, we instantiate virtual environments with eight virtual machines and evaluate the IOPS throughput under 12 test cases across three distinct I/O patterns. The experimental results, depicted in Fig. 4, demonstrate that FairNVMe exhibits superior behavior in the majority of scenarios when compared to other virtualization methods. In scenarios characterized by single-threaded, non-intensive workloads (e.g., n1d1), where ample CPU and NVMe disk resources are available, the throughput differential between FairNVMe and alternative virtualization mechanisms is relatively modest. As the workload intensity and queue depth escalate, the efficiency of FairNVMe becomes more pronounced, particularly under multi-threaded, intensive I/O loads (e.g., n4d32), where resource contention becomes a critical factor. Specifically, FairNVMe enhances the IOPS by up to 3.02x, 23.5%, and 31.2% compared with Virtio, SPDK, and LPNS, respectively. This phenomenon can be attributed to the state-driven fairness controller employed by FairNVMe, which ensures the appropriate allocation of physical queue resources based on the private state of each tenant and alleviates contention among multiple virtual machines.

(2) Average Latency

As illustrated in Fig. 5, with higher workload intensity, the scarcity of physical resources becomes more pronounced, resulting in prolonged response time for I/O requests. As a universal I/O virtualization interface with no specific optimization for NVMe, Virtio gains poor performance in average latency. To mitigate I/O latency, SPDK introduces a zero-copy approach that avoids data replication between user space and kernel space. Additionally, it employs user space drivers instead of kernel drivers, thereby minimizing the frequent context switching between them. However, its static I/O resource allocation strategy makes it less suitable for scenarios that require flexible and dynamic resource management, as resources are allocated prior to application runtime. While LPNS enhances latency predictability through self-feedback mechanisms, it ignores the resource contention for shared NVMe devices among multiple virtual machines, which can lead to potential degradation. In contrast, FairNVMe outperforms Virtio, SPDK, and LPNS, achieving up to 91.9%, 22.1%, and 24.5% latency optimization, respectively. This superior performance is attributed to FairNVMe's budget compensation strategy, which dynamically adjusts physical I/O queue resource allocation according to the requirements of individual VMs, effectively reducing queuing delays and request processing times.

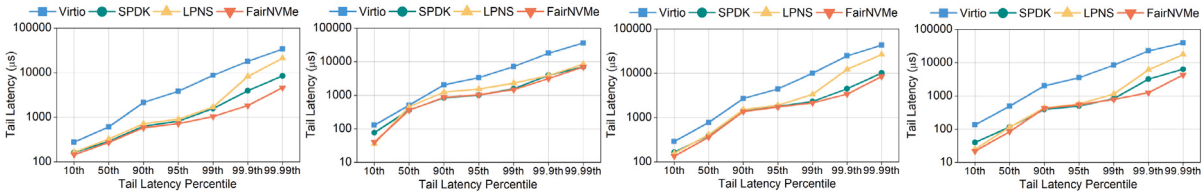
(3) Tail Latency

Fig. 6 provides a comprehensive evaluation of tail latency at various percentiles across four scenarios, including random read, random write, and mixed random read and write operations. The latency results are presented using a logarithmic scale for enhanced clarity, with both the number of jobs and queue depths set to 4. Tail latency is primarily determined by the slowest operations, which directly impact user experience due to delayed responses. The experimental results reveal a trend similar to that observed in average latency measurements. Specifically, FairNVMe reduces the tail latency by up to 94.5%, 61.2%, and 79.5% compared with the other three methods, respectively. These results underscore FairNVMe's ability to substantially reduce the tail



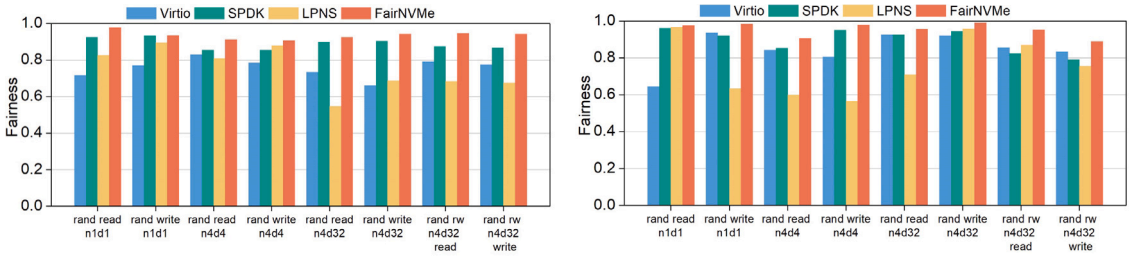
(a) I/O scarce (b) I/O moderate (c) I/O intensive

Fig. 5. The average latency under 12 test cases across three distinct I/O patterns.



(a) Random read (b) Random write (c) Mixed random read 70% (d) Mixed random write 30%

Fig. 6. Tail latency distribution across four scenarios with a logarithmic scale coordinate of latency results.



(a) Intel P3520 (b) Samsung PM1735

Fig. 7. Fairness index based on Eq. (2), indicating the ratio of the minimum to maximum slowdowns (higher values indicate better fairness).

latency, particularly for the 99.9th percentile, which is critical for maintaining high-quality service in cloud environments. Through fair resource allocation, FairNVMe optimizes the processing of the slowest operations and minimizes request delays, thereby enhancing the overall effectiveness and user experience, making it a superior choice in cloud scenarios where both performance and fairness are critical considerations.

5.3. Fairness evaluation

In this subsection, we present a detailed analysis of system fairness evaluation, focusing on three key metrics: the fairness index, the maximum slowdown caused by mutual interference, and the system stability under contention. To comprehensively evaluate FairNVMe's fairness behavior, we conduct a series of experiments on both Intel P3520 and Samsung PM1735.

(1) Fairness Index

Fig. 7 illustrates the fairness results calculated based on Eq. (2), which represents the ratio of the minimum to the maximum slowdowns experienced by VMs. The fairness index ranges from 0 to 1, with higher values indicating better fairness. FairNVMe improves the fairness index across eight test cases, outperforming Virtio, SPDK, and LPNS by up to 51.4%, 15.6%, and 73.2%, respectively. This improvement highlights FairNVMe's capability to enable fair resource allocation among multiple tenants, reducing disparities in performance slowdowns.

(2) Maximum Slowdown

We also evaluate the maximum slowdown caused by mutual interference across various workload patterns. As depicted in Figs. 8 and 9, the system experiences significant slowdowns, with values reaching up to 96x under certain conditions. Despite SPDK and LPNS mitigating resource contention and performance degradation to some extent, they still encounter notable slowdowns. In contrast, with efficient fairness control mechanisms, FairNVMe effectively manages mutual interference, reducing the maximum slowdown value by up to 92.1%, 59.2%, and 58.2% compared with Virtio, SPDK, and LPNS, respectively.

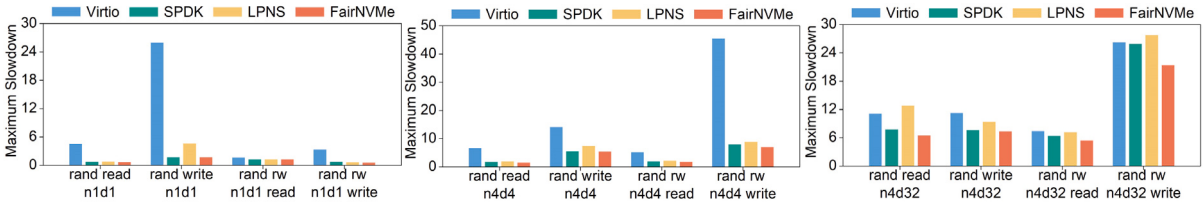
(3) System Stability

To further demonstrate the stability of FairNVMe, we measure the standard deviation (STDEV) of latency behavior, which quantifies the variability and consistency in system performance. With the increase in workload intensity, resource contention among multiple tenants exacerbates, leading to noticeable latency fluctuation and increased STDEV values. As demonstrated in Figs. 10 and 11, FairNVMe provides stable request latency for each tenant, reducing the STDEV value by up to 95.1%, 73.7%, and 91.4% compared to other methods. Through fair I/O queue scheduling, FairNVMe provides a high level of system stability and ensures reliable performance in multi-tenant cloud environments, where consistent latency is essential for sustaining system efficiency and responsiveness.

5.4. Real-world application

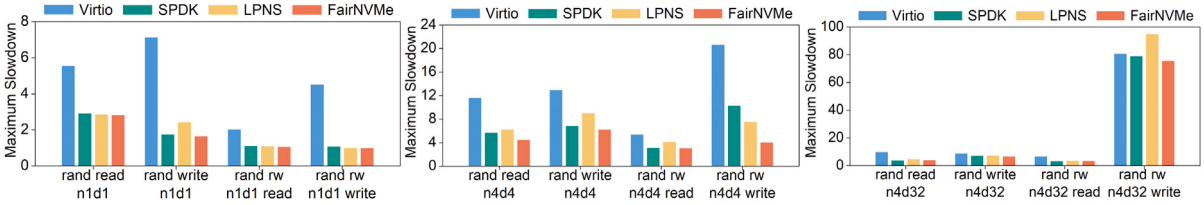
(1) YCSB Benchmarks

To evaluate system efficiency under real-world application scenarios, we utilize the YCSB benchmark [49] to generate key–value store workloads on RocksDB databases. We deploy six guest machines running standard YCSB workloads, ranging from $workload a$ to $workload f$. Each workload corresponds to different combinations of read-to-write ratios and access patterns, reflecting distinct use cases in database



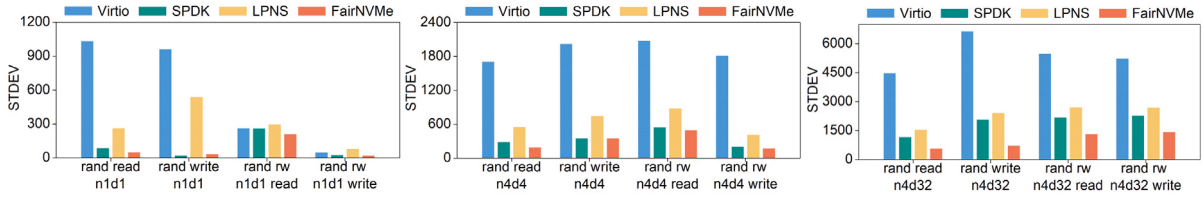
(a) I/O scarce (b) I/O moderate (c) I/O intensive

Fig. 8. Maximum slowdown caused by mutual interference on Intel P3520 (lower values indicate superior outcomes).



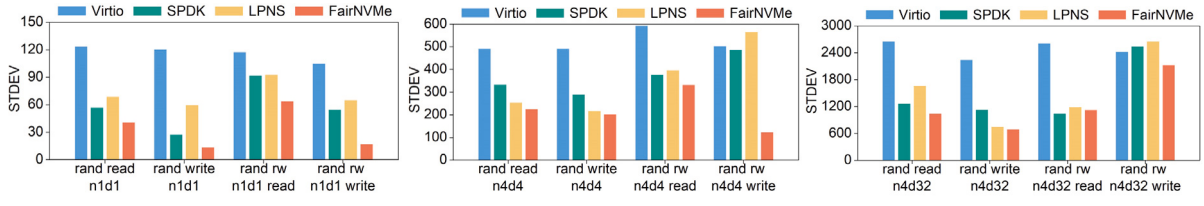
(a) I/O scarce (b) I/O moderate (c) I/O intensive

Fig. 9. Maximum slowdown caused by mutual interference on Samsung PM1735 (lower values indicate superior outcomes).



(a) I/O scarce (b) I/O moderate (c) I/O intensive

Fig. 10. Standard deviation of latency results on Intel P3520 (lower values indicate better stability).



(a) I/O scarce (b) I/O moderate (c) I/O intensive

Fig. 11. Standard deviation of latency results on Samsung PM1735 (lower values indicate better stability).

systems. To simulate resource contention scenarios, we intensify the workload by increasing the *recordcount* and *operationcount* parameters, which result in larger data volumes and higher operation rates, respectively. Fig. 12 illustrates the average latency and corresponding slowdown observed across YCSB workloads. As depicted in Fig. 12(a), FairNVMe exhibits superior latency results on the YCSB benchmarks, achieving average latency reductions of up to 61.0%, 51.4%, and 73.0% compared to Virtio, SPDK, and LPNS, respectively. Additionally, by leveraging state-driven fairness control strategies, FairNVMe effectively mitigates slowdowns and enhances system fairness in real-world scenarios.

(2) Testing under Extreme Workloads

To further evaluate FairNVMe's responsiveness under extreme scenarios, we deploy four tenants, each exhibiting significant disparities and heterogeneous configurations in workload characteristics. Two of these tenants execute specific application workloads generated by Filebench [34], including a webserver and a fileserver. The webserver workload simulates a typical web server environment, focusing on server behavior when handling numerous small files, while the fileserver primarily tests system efficiency when dealing with large files. Following the configuration in FinNVMe [14], the number of threads is set to eight. The remaining two tenants are assigned extreme FIO workloads with distinct characteristics. One tenant executes an I/O scarce workload that employs random read patterns (configured with numjobs = 1 and iodepth = 1), imposing minimal resource demands on the system. The other tenant is dedicated to an I/O intensive workload involving a random write pattern with high concurrency and throughput requirements (configured with numjobs = 8 and iodepth = 32). Fig. 13 presents the normalized p99.9 tail latency results and system fairness metrics. Without flexible I/O scheduling and fairness control, Virtio, SPDK, and LPNS fail to effectively manage the disparities in workload demands, leading to suboptimal resource allocation and higher latency. In contrast, FairNVMe maintains high performance and enhances system fairness even under extreme and heterogeneous workload situations.

5.5. Scalability

To evaluate the scalability of FairNVMe in multi-tenant environments, we expand the number of VMs from 1 to 16 and measure the

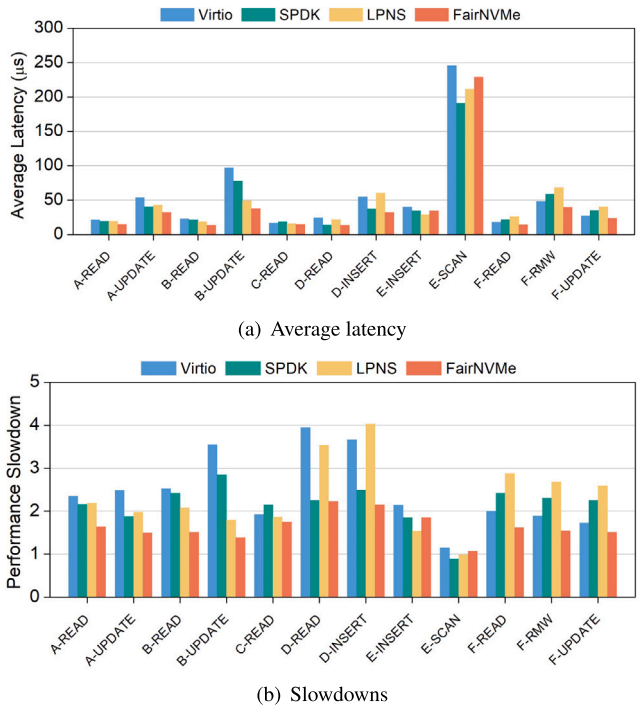


Fig. 12. The average latency and corresponding slowdowns observed in YCSB benchmark evaluations.

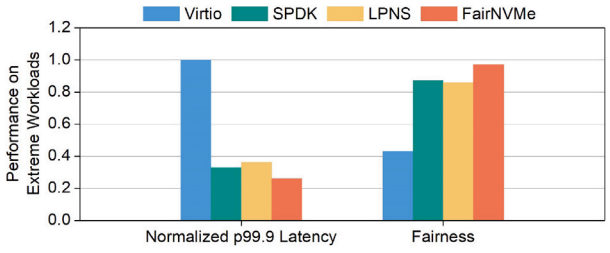


Fig. 13. The tail latency distribution and system fairness assessment under extreme workload scenarios.

overall system IOPS across all tenants. As illustrated in Fig. 14, we assess the IOPS of FairNVMe across various test cases. Under conditions with limited I/O demands, characterized by a small number of jobs and queue depths, the available capacity is not fully utilized. Consequently, deploying additional VMs in these scenarios enhances the utilization of queue resources, leading to a noticeable increase in overall IOPS. As the load intensity rises, system throughput begins to stabilize due to the saturation of I/O queue resources. These findings indicate that FairNVMe efficiently manages the available resources, allowing multi-VMs to operate concurrently without considerable degradation.

Furthermore, we evaluate the IOPS distribution across multiple VMs and the system fairness metrics under intensive random read I/O workloads. As depicted in Fig. 15, when increasing the number of VMs from 1 to 16, the throughput remains evenly distributed across all VMs. FairNVMe prevents any single VM from dominating the I/O queue resources, sustaining consistent and predictable performance for all tenants. Moreover, although the increase in the number of tenants leads to intensified resource contention and a decline in fairness metrics, FairNVMe consistently upholds system fairness, maintaining a fairness metric value of at least 0.905 even as the number of VMs scales to 16. In conclusion, FairNVMe exhibits excellent scalability by enabling fairness control and near-native performance across multiple tenants, which is crucial for the cloud service provider seeking to maximize resource

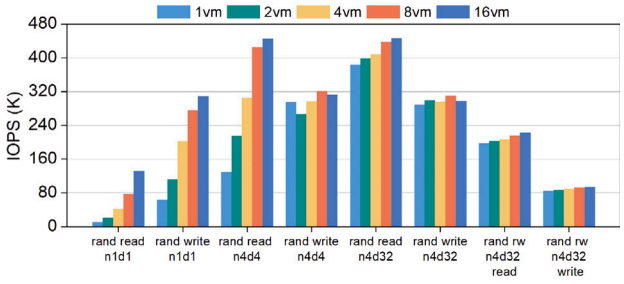


Fig. 14. Scalability of FairNVMe in multi-tenant environments.

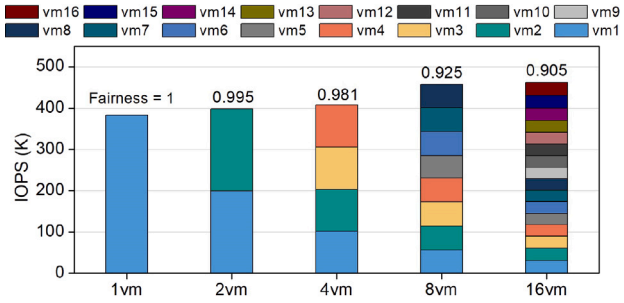


Fig. 15. IOPS distribution of FairNVMe among multiple VMs.

utilization while guaranteeing fairness and stability in multi-tenant cloud environments.

5.6. Workload diversity evaluation

As the former evaluation primarily focuses on 4K random read/write operations, in this subsection, we expand workload diversity by incorporating sequential I/O patterns and variable block sizes to better reflect real-world scenarios. As illustrated in Fig. 16, we evaluate the average latency and maximum performance slowdown across seven distinct I/O block sizes (ranging from 512B to 64K) under intensive sequential read and random write workloads. The average latency results are presented using a logarithmic scale to effectively capture the wide range of latency values. Our experimental analysis reveals that average latency exhibits a slight decrease as block size increases from 512B to 4K, primarily due to improved I/O queue utilization at small block sizes. Furthermore, since most file systems and SSD hardware devices are typically configured with a default block size of 4K, system latency exhibits a decreasing trend when the block size of I/O requests aligns with the 4K boundary. However, we observe a progressive increase in latency from 4K to 64K, which can be attributed to the inherent overhead associated with I/O block separations and the intensified contention in queue resource allocation under larger block sizes. Despite these challenges, our experimental results demonstrate that FairNVMe consistently maintains robust latency performance across various block sizes and diverse I/O patterns, including both sequential and random workloads. Moreover, FairNVMe effectively mitigates performance degradation, validating its adaptability to real-world I/O scenarios.

6. Related work

NVMe Virtualization. As a universal Linux I/O para virtualization framework, Virtio [9] offers a unified interface for various I/O devices. Without specific optimization for NVMe storage devices, it fails to accommodate the unique characteristics of modern high-speed storage technologies. VFIO [29] employs direct pass-through techniques and

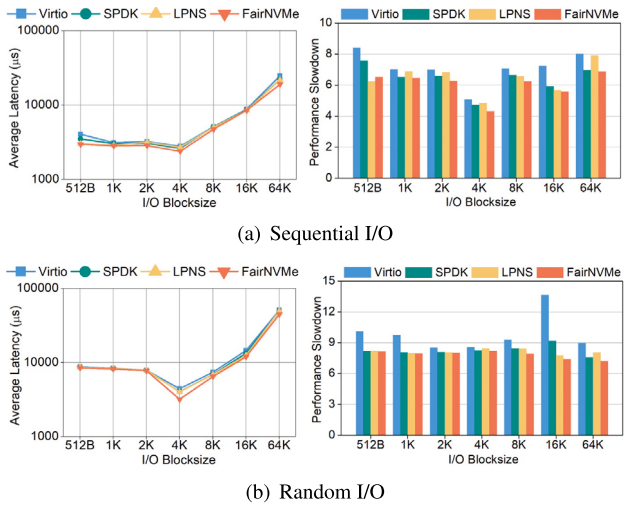


Fig. 16. The average latency and maximum performance slowdown across distinct I/O block sizes under sequential and random I/O patterns.

allocates NVMe devices to individual VMs. Guests can access hardware devices directly via the VFIO-PCI driver, attaining nearly 95% of native results, but the direct passthrough approach hinders device sharing among multiple virtual machines. Fam Zheng [11] presents a userspace NVMe driver based on QEMU, running modified NVMe drivers within the Linux userspace. While Fam Zheng streamlines the software stack involved in I/O request handling and delivers enhanced NVMe virtualization services, the adoption of QEMU-based userspace drivers introduces certain challenges. In particular, the necessity to trap and simulate privileged operations, such as device management and creation, imposes additional software processing overhead and exacerbates tail latency issues. SPDK [12] employs polling mode to mitigate degradation and high latency caused by interrupt-driven mechanisms. By leveraging lock-free queues for message delivery, SPDK optimizes NVMe efficiency, thereby ensuring efficient NVMe virtualization and facilitating device-sharing capabilities. However, SPDK's dependence on Hugepage memory introduces substantial memory resource pressure on physical servers and may limit scalability for the multi-tenant cloud. Mdev-NVMe [13] presents an innovative NVMe storage virtualization mechanism with mediated pass-through, using active polling mode for efficient queue handling. To achieve workload-aware management and latency-predictable QoS control, Peng et al. further propose FinNVMe [14] and LPNS [15] to optimize the performance of mediated pass-through NVMe virtualization, offering enhanced throughput and processing efficiency. Although the aforementioned techniques facilitate the fundamental use of NVMe device virtualization, they overlook the critical fairness requirements in scenarios involving multiple tenants.

Fairness Control. To address the fairness requirements of SSD devices, researchers have explored various solutions [44,51,52]. For instance, FIOS [44] proposes a flash I/O scheduler integrating timeslice management, enabling timeslice fragmentation and concurrent request issuance. Nevertheless, adopting timeslice-based I/O schedulers may result in time slice waste and malicious occupation, leading to poor responsiveness. To eliminate such unresponsiveness, FlashFQ [52] estimates the process of each running flow based on the request costs, and presents a throttled dispatch technique to throttle flows that significantly outpace others to enhance fairness. Moreover, approaches like Fuzzy [27] and FLIN [26] delve into device-level I/O scheduling. Fuzzy [27] introduces a fuzzy logic-based fairness control policy, assigning priority levels to workloads based on their flow intensity. It assigns higher priority to low-intensity workloads since they are more prone to slowdown. FLIN [26] proposes a lightweight transaction scheduler for modern multi-queue SSDs. FLIN identifies four major sources of interference contributing to unfairness and designs a three-stage scheduling algorithm to mitigate interference while adhering to the application-level priorities designated by the host. However, its implementation within the SSD controller firmware escalates the complexity and implementation costs. Different from the above-mentioned methods, FairNVMe emphasizes fairness enhancement in NVMe virtualization and adopts a more flexible approach by implementing fairness mechanisms at the OS level, thereby reducing implementation complexity and enhancing its adaptability to diverse cloud scenarios.

7. Conclusion

In this paper, we introduce FairNVMe, a novel NVMe virtualization solution designed to facilitate effective fairness control in multi-tenant cloud storage systems. To guarantee fair I/O queue scheduling, FairNVMe categorizes tenant status into four distinct levels and generates control signals upon detecting unbalanced or saturated states among tenants, guiding the system to trigger dynamic budget compensation based on the estimated request costs. Beyond its technical contributions, FairNVMe holds broader implications for cloud service providers, particularly in meeting stringent Service Level Agreements (SLAs). By delivering consistent and predictable I/O performance, FairNVMe enhances QoS for tenants, which is critical in dense cloud environments where competition for resources is often intense. Furthermore, the core principles of FairNVMe, such as state-driven resource management and dynamic budget compensation, can be extended to other resource allocation challenges in cloud computing, offering a pathway to more efficient and equitable resource utilization.

CRediT authorship contribution statement

Zhaoyang Huang: Writing – original draft, Methodology, Conceptualization. **Yifu Zhu:** Software, Investigation. **Xin Kuang:** Validation, Formal analysis. **Yanjie Tan:** Visualization, Data curation. **Huailiang Tan:** Resources, Project administration. **Keqin Li:** Supervision.

Declaration of competing interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Data availability

The data that has been used is confidential.

References

[1] A. Ali, Ö. Özkasap, Spatial and thermal aware methods for efficient workload management in distributed data centers, Future Gener. Comput. Syst. 153 (2024) 360–374.
[2] S.B. Nath, S.K. Addya, S. Chakraborty, S.K. Ghosh, CSMD: Container state management for deployment in cloud data centers, Future Gener. Comput. Syst. 162 (2025) 107495.
[3] Z. Jia, J. Zhan, L. Wang, C. Luo, W. Gao, Y. Jin, R. Han, L. Zhang, Understanding big data analytics workloads on modern processors, IEEE Trans. Parallel Distrib. Syst. 28 (6) (2016) 1797–1810.
[4] Nvmepress, NVMe specifications, 2021, http://www.nvmexpress.org/specifications/.
[5] L.A. Barroso, J. Clidaras, The datacenter as a computer: An introduction to the design of warehouse-scale machines, Springer Nature, 2022.
[6] G. Ananthanarayanan, A. Ghodsi, S. Shenker, I. Stoica, {Disk-locality} in datacenter computing considered irrelevant, in: 13th Workshop on Hot Topics in Operating Systems (HotOS XIII), 2011.
[7] Y. Son, H. Han, H.Y. Yeom, Optimizing file systems for fast storage devices, in: Proceedings of the 8th ACM International Systems and Storage Conference, 2015, pp. 1–6.

[8] B. Jun, D. Shin, Workload-aware budget compensation scheduling for NVMe solid state drives, in: 2015 IEEE Non-Volatile Memory System and Applications Symposium, NVMSA, IEEE, 2015, pp. 1–6.
[9] R. Russell, Virtio: towards a de-facto standard for virtual I/O devices, ACM SIGOPS Oper. Syst. Rev. 42 (5) (2008) 95–103.
[10] Z. Yang, M. Hoseinzadeh, P. Wong, J. Artoux, C. Mayers, D.T. Evans, R.T. Bolt, J. Bhimani, N. Mi, S. Swanson, H-NVMe: A hybrid framework of NVMe-based storage system in cloud computing environment, in: 2017 IEEE 36th International Performance Computing and Communications Conference, IPCCC, IEEE, 2017, pp. 1–8.
[11] F. Zheng, Userspace NVMe driver in QEMU, in: KVM Forum 2017, 2017, pp. 25–27.
[12] Z. Yang, J.R. Harris, B. Walker, D. Verkamp, C. Liu, C. Chang, G. Cao, J. Stern, V. Verma, 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), IEEE, 2017, pp. 154–161.
[13] B. Peng, H. Zhang, J. Yao, Y. Dong, Y. Xu, H. Guan, {Mdev-nvme}: A {nvme} storage virtualization solution with mediated {pass-through}, in: 2018 USENIX Annual Technical Conference (USENIX ATC 18), 2018, pp. 665–676.
[14] B. Peng, M. Yang, J. Yao, H. Guan, A throughput-oriented nvme storage virtualization with workload-aware management, IEEE Trans. Comput. 70 (12) (2020) 2112–2124.
[15] B. Peng, C. Guo, J. Yao, H. Guan, {LpNS}: Scalable and {latency-predictable} local storage virtualization for unpredictable {nvme}{SSDs} in clouds, in: 2023 USENIX Annual Technical Conference (USENIX ATC 23), 2023, pp. 785–800.
[16] Y. Dong, X. Yang, J. Li, G. Liao, K. Tian, H. Guan, High performance network virtualization with SR-IOV, J. Parallel Distrib. Comput. 72 (11) (2012) 1471–1480.
[17] H. Li, M. Hao, S. Novakovic, V. Gogte, S. Govindan, D.R. Ports, I. Zhang, R. Bianchini, H.S. Gunawi, A. 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, 2020, pp. 591–605.
[18] D. Kwon, J. Boo, D. Kim, J. Kim, {FVM}:{fpga-assisted} virtual device emulation for fast, scalable, and flexible storage virtualization, in: 14th USENIX Symposium on Operating Systems Design and Implementation (OSDI 20), 2020, pp. 955–971.
[19] J. Zhang, M. Kwon, D. Gouk, S. Koh, C. Lee, M. Alian, M. Chun, M.T. Kandemir, N.S. Kim, J. Kim, et al., {FlashShare}: Punching through server storage stack from kernel to firmware for {ultra-low} latency {SSDs}, in: 13th USENIX Symposium on Operating Systems Design and Implementation (OSDI 18), 2018, pp. 477–492.
[20] C. Ji, L. Wang, Q. Li, C. Gao, L. Shi, C.-L. Yang, C.J. Xue, Fair down to the device: A GC-aware fair scheduler for SSD, in: 2019 IEEE Non-Volatile Memory Systems and Applications Symposium, NVMSA, IEEE, 2019, pp. 1–6.
[21] C.-H. Wu, L.-T. Chen, R.-J. Hsu, J.-Y. Dai, A state-aware method for flows with fairness on NVMe SSDs with load balance, IEEE Trans. Cloud Comput. 11 (3) (2023) 3040–3054.
[22] S. Metin, C. Özturan, Quantised and simulated max–min fairness in blockchain ecosystems, Future Gener. Comput. Syst. 151 (2024) 260–271.
[23] L.K. Organization, CFQ (complete fairness queueing), 2020, https://www.kernel.org/doc/Documentation/block/cfq-iosched.txt.
[24] P. Valente, M. Andreolini, Improving application responsiveness with the bfq disk i/o scheduler, in: Proceedings of the 5th Annual International Systems and Storage Conference, 2012, pp. 1–12.
[25] Axboe, Mq-Deadline: Add blk-mq adaptation of the Deadline IO scheduler, 2021, https://patchwork.kernel.org/patch/9475509/.
[26] A. Tavakkol, M. Sadrosadati, S. Ghose, J. Kim, Y. Luo, Y. Wang, N.M. Ghiasi, L. Orosa, J. Gómez-Luna, O. Mutlu, FLIN: Enabling fairness and enhancing performance in modern NVMe solid state drives, in: 2018 ACM/IEEE 45th Annual International Symposium on Computer Architecture, ISCA, IEEE, 2018, pp. 397–410.
[27] S. Tripathy, D. Sahoo, M. Satpathy, M. Mutyam, Fuzzy fairness controller for NVMe SSDs, in: Proceedings of the 34th ACM International Conference on Supercomputing, 2020, pp. 1–12.
[28] R. Liu, Z. Tan, Y. Shen, L. Long, D. Liu, Fair-ZNS: Enhancing fairness in ZNS SSDs through self-balancing I/O scheduling, in: IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems, 2024, pp. 2012–2022.
[29] A. Williamson, VFIO: A user's perspective, in: KVM Forum, 2012.
[30] Y. Peng, H. Peng, InferFair: Towards qos-aware scheduling for performance isolation guarantee in heterogeneous model serving systems, Future Gener. Comput. Syst. 150 (2024) 10–20.
[31] F. Xu, F. Liu, H. Jin, Heterogeneity and interference-aware virtual machine provisioning for predictable performance in the cloud, IEEE Trans. Comput. 65 (8) (2015) 2470–2483.
[32] S.A. Murad, Z.R.M. Azmi, A.J.M. Muzahid, M.K.B. Bhuiyan, M. Saib, N. Rahimi, N.J. Prottasha, A.K. Bairagi, SG-PBFS: Shortest gap-priority based fair scheduling technique for job scheduling in cloud environment, Future Gener. Comput. Syst. 150 (2024) 232–242.
[33] R. Liu, Z. Tan, L. Long, Y. Wu, Y. Tan, D. Liu, Improving fairness for SSD devices through DRAM over-provisioning cache management, in: IEEE Transactions on Parallel and Distributed Systems, 2022, pp. 2444–2454.
[34] T. Vasily, Filebench: A flexible framework for file system benchmarking.; login, USENIX Mag. 41 (2016) 6.
[35] S.S. Mousavi Nik, M. Naghibzadeh, Y. Sedaghat, Task replication to improve the reliability of running workflows on the cloud, Clust. Comput. 24 (1) (2021) 343–359.
[36] N. Jia, VFIO mediated devices, 2016, https://www.kernel.org/doc/Documentation/vfio-mediated-device.txt.
[37] A. Gulati, A. Merchant, M. Uysal, P. Padala, P. Varman, Efficient and adaptive proportional share I/O scheduling, ACM SIGMETRICS Perform. Eval. Rev. 37 (2) (2009) 79–80.
[38] H. Tan, L. Huang, Z. He, Y. Lu, X. He, DMVL: An I/O bandwidth dynamic allocation method for virtual networks, J. Netw. Comput. Appl. 39 (2014) 104–116.
[39] J.S. Hunter, The exponentially weighted moving average, J. Qual. Technol. 18 (4) (1986) 203–210.
[40] G. Carlucci, L. De Cicco, S. Holmer, S. Mascolo, Congestion control for web real-time communication, IEEE/ACM Trans. Netw. 25 (5) (2017) 2629–2642.
[41] Y. Hu, H. Jiang, D. Feng, L. Tian, H. Luo, S. Zhang, Performance impact and interplay of SSD parallelism through advanced commands, allocation strategy and data granularity, in: Proceedings of the International Conference on Supercomputing, 2011, pp. 96–107.
[42] Y. Kim, B. Tauras, A. Gupta, B. Urgaonkar, Flashsim: A simulator for nand flash-based solid-state drives, in: 2009 First International Conference on Advances in System Simulation, IEEE, 2009, pp. 125–131.
[43] A. Tavakkol, J. Gómez-Luna, M. Sadrosadati, S. Ghose, O. Mutlu, {Mqsim}: A framework for enabling realistic studies of modern {multi-queue}{sSD} devices, in: 16th USENIX Conference on File and Storage Technologies (FAST 18), 2018, pp. 49–66.
[44] S. Park, K. Shen, FIOS: a fair, efficient flash I/O scheduler., in: FAST, vol. 12, 2012, 13–13.
[45] M. Hedayati, K. Shen, M.L. Scott, M. Marty, {Multi-queue} fair queuing, in: 2019 USENIX Annual Technical Conference (USENIX ATC 19), 2019, pp. 301–314.
[46] J. Coulter, Intel DC P3520 series enterprise nvme ssd, 2020, https://www.tweaktown.com/reviews/8010/intel-dc-p3520-2tb-enterprise-pcie-nvme-ssd-review/index.html.
[47] Samsung, Samsung enterprise SSD PM1735, 2024, https://semiconductor.samsung.com/cn/ssd/enterprise-ssd/pm1733-pm1735/.
[48] J. Axboe, FIO: Flexible IO tester, 2024, https://github.com/axboe/fio.
[49] B.F. Cooper, A. Silberstein, E. Tam, R. Ramakrishnan, R. Sears, Benchmarking cloud serving systems with YCSB, in: Proceedings of the 1st ACM Symposium on Cloud Computing, 2010, pp. 143–154.
[50] Camelab, Flash-based block traces, 2020, http://trace.camelab.org/2016/03/01/flash.html.
[51] Q. Zhang, D. Feng, F. Wang, Y. Xie, An efficient, qos-aware I/O scheduler for solid state drive, in: 2013 IEEE 10th International Conference on High Performance Computing and Communications & 2013 IEEE International Conference on Embedded and Ubiquitous Computing, IEEE, 2013, pp. 1408–1415.
[52] K. Shen, S. Park, {FlashFQ}: A fair queueing {I/O} scheduler for {flash-based}{ssds}, in: 2013 USENIX Annual Technical Conference (USENIX ATC 13), 2013, pp. 67–78.

Zhaoyang Huang received the BS degree in Computer Science and Technology from the China West Normal University in 2020. She is currently working toward the PhD degree in the College of Computer Science and Electronic Engineering at Hunan University. Her current research interests include cloud computing and data centers.

Yifu Zhu received the BS degree in the College of Electronic Science and Engineering from Jilin University in 2019, and the MS degree from Hunan University in 2023. He is currently working toward the PhD degree in the College of Computer Science and Electronic Engineering, Hunan University. His current research interests include FPGA and real-time systems

Xin Kuang received the BS degree in Computer Science and Technology from Jiangxi University of Science and Technology in 2020, and the MS degree from Hunan University in 2023. Her current research interests include GPU Virtualization and Virtual Desktop Infrastructure.

Yanjie Tan is a postdoctor in the College of Computer Science and Electronic Engineering at Hunan University. He received the BS degree and the MS degree from Huazhong University of Science and Technology, China, in 2011 and 2015, and the PhD degree from Hunan University, China, in 2021. His current research interests include real-time system and image and video processing.

Huailiang Tan received the BS degree from Central South University, China, in 1992, and the MS degree from Hunan University, China, in 1995, and the PhD degree from Central South University, China, in 2001. He has more than eight years of industrial research and development experience in the field of information technology. He was a visiting scholar at Virginia Commonwealth University from 2010 to 2011. He is currently a full professor of Computer Science and Technology with Hunan University, China. His research interests include high performance I/O, image and video processing, and embedded systems.

Keqin Li received a B.S. degree in computer science from Tsinghua University in 1985 and a Ph.D. degree in computer science from the University of Houston in 1990. He is a SUNY Distinguished Professor with the State University of New York and a National Distinguished Professor with Hunan University (China). He has authored or co-authored more than 1000 journal articles, book chapters, and refereed conference papers. He received several best paper awards from international conferences including PDPTA-1996, NAECON- 1997, IPDPS-2000, ISPA-2016, NPC-2019, ISPA-2019, and CPSCom-2022. He holds nearly 75 patents announced or authorized by the Chinese National Intellectual Property Administration. He is among the world's top five most influential scientists in parallel and distributed computing in terms of single-year and career-long impacts based on a composite indicator of the Scopus citation database. He was a 2017 recipient of the Albert Nelson Marquis Lifetime Achievement Award for being listed in Marquis Who's Who in Science and Engineering, Who's Who in America, Who's Who in the World, and Who's Who in American Education for over twenty consecutive years. He received the Distinguished Alumnus Award from the Computer Science Department at the University of Houston in 2018. He received the IEEE TCCLD Research Impact Award from the IEEE CS Technical Committee on Cloud Computing in 2022 and the IEEE TCSVC Research Innovation Award from the IEEE CS Technical Community on Services Computing in 2023. He won the IEEE Region 1 Technological Innovation Award (Academic) in 2023. He is a Member of the SUNY Distinguished Academy. He is an AAAS Fellow, an IEEE Fellow, an AAIA Fellow, and an ACIS Founding Fellow. He is an Academician Member and Fellow of the International Artificial Intelligence Industry Alliance. He is a Member of Academia Europaea (Academician of the Academy of Europe).