# Enabling Multi-tenancy on SSDs with Accurate IO Interference Modeling

Lokesh N. Jaliminche*
University of California, Santa Cruz, USA
ljalimin@ucsc.edu

Chandranil (Nil) Chakraborttii
Trinity College, Hartford, USA
nil.chakraborttii@trincoll.edu

Changho Choi
Samsung Semiconductor, Inc, USA
changho.c@samsung.com

Heiner Litz
University of California, Santa Cruz, USA
hlitz@ucsc.edu

## ABSTRACT

Technological advancements in the past decades have substantially increased the capacity and performance of Solid State Drives (SSDs). Provisioning such high-capacity SSDs among tenants can reap multiple benefits, such as elevated performance, efficient resource utilization, and cost savings through reduced Total Cost of Ownership. However, workloads perform poorly when co-located with others on the same SSD due to IO Interference, potentially violating Service Level Objectives (SLOs). High overprovisioning can address the SLO issue, however, it entails low utilization. Prior works proposed Machine Learning (ML) techniques to predict SSD performance in the presence of interfering tenants for optimizing workload placement. However, we find that these works suffer from two notable limitations. First, previous ML models do not capture interference impact due to the non-uniform workload characteristics and SSD internals. Second, they fail to compute interference of an arbitrary number of workloads due to a lack of feature aggregation. As a result, these works still offer low utilization and can only enforce weak SLOs. To address these limitations, we propose a Gray-box feature representation and aggregation technique to capture the IO interference impact of multiple non-uniform workloads based on internal SSD characteristics. Our technique improves prediction accuracy by 12x (lower mean absolute error) over prior works, resulting in up to 60% higher resource utilization or enforcing up to 2.5× stricter SLOs.


*SoCC '23, October 30–November 1, 2023, Santa Cruz, CA, USA*

ACM ISBN 979-8-4007-0387-4/23/11.
https://doi.org/10.1145/3620678.3624657

## CCS CONCEPTS

• **Computing methodologies** → *Modeling methodologies*; • **Information systems** → **Storage management**.

## KEYWORDS

IO Interference, Performance Modeling, Machine Learning, Resource Allocation, SSDs, Performance Isolation

**ACM Reference Format:**
Lokesh N. Jaliminche, Chandranil (Nil) Chakraborttii, Changho Choi, and Heiner Litz. 2023. Enabling Multi-tenancy on SSDs with Accurate IO Interference Modeling. In *ACM Symposium on Cloud Computing (SoCC '23), October 30–November 1, 2023, Santa Cruz, CA, USA.* ACM, New York, NY, USA, 17 pages. https://doi.org/10.1145/3620678.3624657

## 1 BACKGROUND AND INTRODUCTION

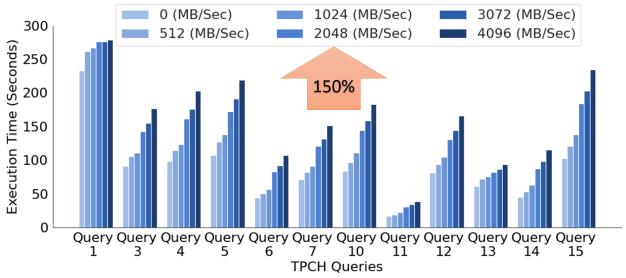


**Figure 1: Postgres performance degradation due to IO Interference**

Modern SSDs support multi-terabyte of storage capacity by exploiting techniques such as planar scaling [74], 3D integration [50], and multi-level cells [49]. This trend seems to persist with SSDs utilizing Host Memory Buffer(HMB) [36]. In addition, SSD manufacturers have also been scaling the IO bandwidth of SSDs by utilizing multiple parallel NAND chips, multiple channels, and multi-plane operations [63]. As a result, single workloads can now rarely utilize the performance and capacity offered by a modern SSD. Public cloud

providers leverage this trend by partitioning physical SSDs into multiple virtual SSDs and leasing them to customers as individual storage resources [37, 38, 48, 70]. This greatly improves resource utilization and cost-effectiveness. However, it also introduces the challenge of IO interference, which can significantly degrade performance for the tenants sharing the SSD. To demonstrate this effect, we conduct an experiment running TPC-H [14] queries on PostgreSQL while concurrently scheduling multiple interfering workloads on the same SSD. Figure 1 shows that, as the IO bandwidth of the Interfering Workloads increases, PostgreSQL suffers significant performance degradation as query execution time increases by up to 150%. This happens due to the internal structure of SSDs, which leads to contention in accessing shared resources (such as flash chips) across workloads. In such cases, tenants' service level objectives (SLO) can be violated, resulting in high over-provisioning, affecting resource utilization and cost-effectiveness [6].

Prior works propose several approaches to address the challenge of IO interference. These approaches can be broadly divided into two categories. The first category involves works suggesting changes to the internal architecture of SSDs [11, 27, 29, 30, 32, 41, 47, 62, 65, 67] isolating physical resources. While these techniques can reduce interference, they are difficult to adapt, and often waste resources due to static partitioning as they can only support a small number of workloads or tenants. Furthermore, partitioning SSD resources reduces the number of effective resources applications can use, limiting performance. The second category of works uses black-box Machine Learning (ML) algorithms to model the device performance in the presence of IO interference and inform resource allocation strategies improving efficiency and utilization while maintaining adequate performance [12, 13, 15, 52]. These approaches are easy to adapt as they do not require changes to SSD architecture. Since they do not isolate the physical resources, applications can utilize all the resources available, improving overall performance and utilization.

Unfortunately, determining IO interference is non-trivial due to the large variety of workload characteristics and internal SSD architectures. In particular, the workload characteristics affecting IO interference include the read and write intensity, IO size, IO alignment, access patterns, and burstiness. At the same time, SSDs respond differently to such workload characteristics based on their internal organization of channels, flash chips, and their implemented allocation policies. Consequently, prior works [12, 13, 15, 52] suffer from the following shortcomings.

First, they are limited to predicting workload interference between two or, at most, a small, fixed number of workloads. Multiple workloads cannot be supported easily as their ML models cannot simultaneously accept an arbitrary number of feature sets (one for each workload). Second, prior works are incapable of extracting complex behaviors such as burstiness from an observed workload as they consider only a limited feature set, such as IO size, IO intensity, and IO access patterns (sequential and random). Third, because black-box prediction models lack the understanding of an SSD's internal device architecture, without extensive feature engineering, they cannot determine the behavior of a particular architecture for a given workload.

We address these challenges with a novel gray-box feature representation and aggregation technique. We first perform a detailed quantitative analysis of workload characteristics to determine their precise effects on IO interference. Then, we propose new features that can accurately represent the non-uniform nature of workloads (such as variability in IO size and burstiness of the workloads) and their interference impact on SSD's performance. We refer to our technique as gray-box, as we relate each of our observations to the device architecture. For instance, we find that for our analyzed SSDs, the interference impact of IO size can be captured by categorizing them according to their alignment with the page size of the SSD. Furthermore, we introduce a new feature aggregation technique that enables interference prediction across an *arbitrary* number of workloads by aggregating their features to represent a single aggregated workload. In summary, we make the following contributions.

**(1)** A root cause analysis of SSD internals and workload characteristics for modeling IO interference (Section3).

**(2)** A novel feature representation and aggregation technique for representing and aggregating IO workload characteristics enabling multi-workload interference representation with its usage model (Section 4, 5).

**(3)** An evaluation on several real-world workload traces (Alibaba [42] and Tencent [75]), outperforming prior works by up to 12×(lower mean absolute error) (Section 6).

**(4)** Demonstrating increased resource utilization and reduced TCO by reducing the number of SSDs for workload placement (Section 6.2).

## 2 SSD INTERNALS ANALYSIS

In this section, we briefly introduce the internal architecture of modern SSDs and then highlight specific SSD properties that cause IO interference. Based on these observations, Section 4 will introduce the necessary features to support our machine-learning mechanism.

### 2.1 Internal Device Parallelism

Figure 2 shows the basic architecture of a modern SSD consisting of a controller, DRAM, and multiple flash chips. Multiple flash chips are connected to the controller via channels, while each flash chip consists of planes, blocks, and pages. SSDs leverage parallel flash chips and channels to boost

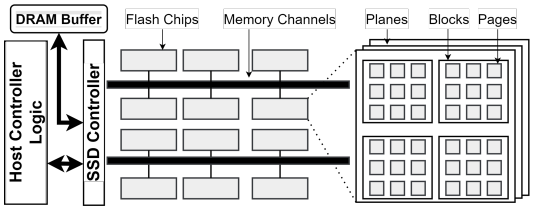


**Figure 2: Basic SSD Architecture**

performance [1, 23–26, 56]. However, concurrent usage of hardware resources (e.g., memory channels) introduces contention, causing performance degradation[25, 41]. In section 3.2 and 3.3, we analyze how such contention impacts IO interference.

As flash memory cannot be written without a prior erase operation, SSDs utilize out-of-place updates and sophisticated allocation policies to distribute write traffic equally across flash chips. Consequently, spatial address patterns such as sequential or random accesses do not necessarily determine the physical location of a sector. In contrast, reads cannot be load-balanced across flash chips as the physical location is determined on each write operation. We study how such spatial IO access patterns affect interference in section 3.5

## 2.2 Read and Write Amplification

Read respectively write amplification is defined as the ratio of data accessed from the SSD and the data requested by software [72]. Such read/write amplification happens due to garbage collection [22] but also because of unaligned IO sizes and unaligned IO offsets [31, 33, 44, 72].

**Unaligned IO Size operations** occur if reads or writes access an amount of data that is not a multiple of the page size (4K). Particularly, if write operations are smaller than the page size, a read-modify-write (RMW) operation must be performed. Similarly, when read operations are smaller than the page size, the entire page is fetched into a read buffer, and the required data is subsequently extracted from the relevant offset. In both scenarios, an excessive amount of data is read and written, causing read or write amplification. We will analyze this effect in section 3.1.

**Unaligned IO offset Operations** occur if reads or writes utilize addresses that are not page-aligned, causing RMW operations on multiple pages, significantly increasing read and write amplification compared to aligned offset operations [72]. We analyze the impact on IO interference of unaligned IO offsets in section 3.6.

## 2.3 Read Write Assymetry

NAND flash writes require high voltages to re-program the floating gate transistors, taking substantially longer than reads [46, 69]. Flash chips only support one outstanding operation (read, write, or erase) at a time. As a result, a heavy write workload can reduce the overall read performance. In section 3.4, we evaluate how read-write asymmetry affects the performance of neighboring workloads.

# 3 WORKLOAD CHARACTERISTICS ANALYSIS

This section analyzes the effect of specific IO workload properties on interference. We are particularly interested in investigating non-linear effects as their representation requires more sophisticated IO interference modeling. We use this analysis as a guide for proposing our SSD-specific feature representation. In particular, we analyze the following workload properties:

- IO Size
- IO Rate
- IO Depth
- IO Type
- Temporal and Spatial IO access Pattern

To enable a thorough understanding of each of the above properties corresponding to SSD internals, we perform our experiments with two types of SSDs, shown in Table 1. We utilize the flexible IO tester (FIO) [3] to develop synthetic benchmarks where we define interferering workloads (IW) and generally one workload under test (WUT). We run these benchmarks and observe the performance degradation for the WUT. For each benchmark, we vary one of the above properties for IW, keeping all the other properties constant, allowing us to understand the IO interference impact of that specific property. We use a lower IO rate for the SATA SSD compared to the NVMe SSD in line with their maximum supported bandwidth. We describe the FIO features used in each benchmark in their respective sections in a tabular format.

We use libaio [16] as the IO engine and use 8 threads to generate each IO workload for all the experiments. To analyze the IO interference impact caused by a particular IO metric, we use the delta between the IO bandwidth degradation observed by the WUT (y-axis) corresponding to variation in that IO metric (x-axis). To calculate the difference (Delta), we first calculate the average IO bandwidth degradation by capturing the WUT's average IO bandwidth running in isolation and then compare it against the bandwidth when exposed to IO interference (Equation 1). Then we calculate the Delta between the two corresponding benchmarks to observe the variation in the IO interference impact corresponding to different values of a particular IO metric (Equation 2). This applies to all the experiments except for section 3.2 and 3.3, where we evaluate the average bandwidth degradation for WUT when run against an IW.

| Hardware/Software | Configuration |
|---|---|
| CPU | Intel Xeon @ 2.00GHz,<br>2 Sockets, 14 Cores,<br>2 Threads per core, |
| DRAM | 80GB |
| SSD 1 (NVMe) | Samsung PM1735 (3.4TB)<br>Peak Read BW: ≈6GB/s<br>Peak Write BW: ≈3GB/s<br>Peak Mixed BW: ≈3GB/s |
| SSD 2 (SATA) | Samsung 870 EVO (1.8TB)<br>Peak Read BW: ≈512MB/s<br>Peak Write BW: ≈512MB/s<br>Peak Mixed BW: ≈300-512MB/s |
| Benchmarking Tool | FIO |
| Operating System | Ubuntu 20.04.4 LTS |

**Table 1: System Setup for Experiments**

Due to space constraints, our analysis primarily focuses on the read metrics, omitting the write sensitivity analysis. While write performance is important for tasks such as write-ahead-logging, modern SSDs generally complete writes as soon as they are absorbed by the battery-backed DRAM write buffer and DRAM is less susceptible to interference than flash. Nevertheless, as we will show in section 6, our feature representation and aggregation technique (discussed in section 4) takes into account the impact of IO interference on both read and write workloads.

$$\text{Bandwidth Degradation} = \frac{\text{Non-Interference BW} - \text{Interference BW}}{\text{Non-Interference BW}} \times 100 \quad (1)$$

$$\text{Delta Bandwidth Degradation} = |\text{BW Degradation(A)} - \text{BW Degradation(B)}| \quad (2)$$

## 3.1 IO Size Misalignment

| Workload | FIO parameters<br>(IO type, IO size, IO rate, IO depth) |
|---|---|
| WUT(NVMe) | randread,4k,524288,128 (2GB/Sec) |
| WUT(SATA) | randread, 4k,65536,128 (0.25GB/Sec) |
| IW(NVMe) | randwrite,4K:8K,349520,128(2GB/Sec)<br>randwrite,6k,349520,128(2GB/Sec) |
| IW(SATA) | randwrite,4K:8K,43680,128(0.25GB/Sec)<br>randwrite,6k,43680,128(0.25GB/Sec) |

**Table 2: IO Size Sensitivity Experiment**

To analyze the IO size impact on IO interference, we define several IWs sweeping their IO sizes from 4 KB to 1024 KB. It is generally accepted that larger IO sizes have a higher IO interference impact since a higher amount of data for IO occupies more SSD resources. So we focus on comparing the IO interference impact between the aligned vs. unaligned IO sizes corresponding to the page size of the SSD (4KB). For a fair comparison, we use similar IO bandwidth across all the IWs. To keep the IO rate and depth constant, we use two 4KB aligned sizes whose average results in 4KB unaligned size. This allows us to ensure that only the IO size property of the IWs is varied. For instance, Table 2 shows the configuration of IWs where we use two page-aligned IO sizes of 4KB and 8KB that achieves the IO bandwidth of 2GB/Sec (each IO size is responsible for 50% of the total IO rate). For the corresponding page-unaligned workload, we use a 6KB IO size that has exactly same IO rate and IO depth. The WUT uses a fixed size of 4KB across all the IWs.

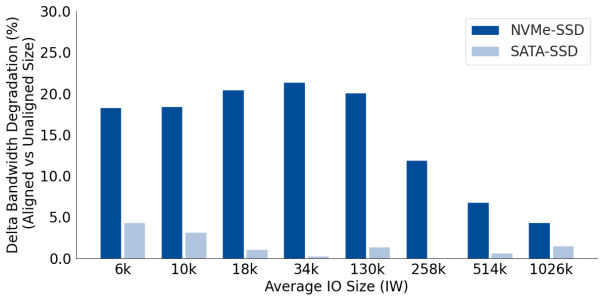


**Figure 3: IO Size Misalignment Sensitivity**

From Figure 3, we observe that for NVMe SSD, the delta between bandwidth degradation caused by aligned and unaligned size workloads is ≈ 20%, which decreases as IO sizes increase. This stark difference in performance degradation caused by aligned and unaligned IO size is explained by the read amplification and write amplification issue explored in section 2.2. This delta keeps decreasing for higher IO sizes because an increase in the IO size increases the number of aligned writes vs. unaligned writes. For instance, a 6 KB IO size causes one aligned write and one unaligned write. However, 1026KB causes 256 aligned writes and one unaligned write reducing the IO interference impact of unaligned writes. However, for SATA SSD, we do not see a significant difference in the IO interference impact of aligned vs unaligned writes even for smaller writes. We attribute this effect to the data transfer latency of the SATA interface, which is significantly higher than the NVMe interface [66], hiding the impact of unaligned writes in SSDs.

## 3.2 IO Rate

To understand IO interference sensitivity to the IO rate, we devise a single WUT and several IWs sweeping only the IO

rate (Table 3). We only discuss the results for the NVMe SSD as our observations apply equally to the SATA SSD. From Figure 4, we can see that the IO interference impact of IW keeps increasing from 0 to ≈ 30% as we increase the IO rate of the interfering workload. We see this effect as an increase in the IO rate increases the contention for accessing the internal SSD resources (Section 2.1), such as memory channels and flash chips, consequently affecting the performance of the WUT. However, the IO interference impact is similar for the IO rate from 314k to 524k (≈ 15–18%). Such behavior can arise from the correlation between the cost of IO operations and the IO rate. For instance, while contending for the IO resources with similar IO depth, because less expensive IO requests get served faster, such workloads can maintain a higher IO rate, especially when contending with larger IO sizes workloads. As a result, they suffer lesser IO interference. This effect can change with the IO depth, which we will discuss in section 3.3

| **Workload** | **FIO parameters** (IO type, IO size, IO rate, IO depth) |
|---|---|
| WUT (NVMe) | randread,4k, 524288,128 (2GB/Sec) |
| WUT (SATA) | randread,4k, 65536,128 (0.25GB/Sec) |
| IW (NVMe) | randwrite,4k, 104k - 629K, 128 (0.4 - 2.4GB/Sec) |
| IW (SATA) | randwrite,4k, 4k - 65K, 128 (0.02 - 0.25GB/Sec) |

**Table 3: IO rate sensitivity experiment**

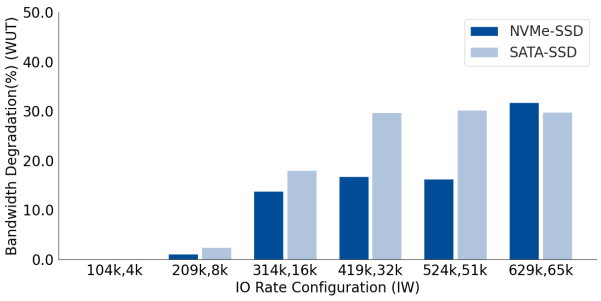


**Figure 4: IO rate sensitivity**

## 3.3 IO Depth

To understand the effect of IO depth, we design an experiment where we sweep the IO depth of the IWs from 8 to 16384, keeping IOPS constant (shown in Table 4). We only discuss the results for the NVMe SSD as our observations apply equally to the SATA SSD. Figure 5 shows that as we increase the IO depth, WUT suffers higher bandwidth degradation, increasing from 0 to 30%. We see such effect due to increased contention to access the SSD resources corresponding to IO depth. However, we see a step function where IO depths 64-256 and 8192-16384 have similar IO interference impacts. We see such behavior because IO depth generally helps to achieve certain required IO bandwidth. Once the required bandwidth of the workload is already met, further increases in IO depth do not cause higher IO interference, which is the case for the IO depths 8192-16384.

| **Workload** | **FIO parameters** (IO type, IO size, IO rate, IO depth) |
|---|---|
| WUT (NVMe) | randread, 4k, 524288, 128(2GB/Sec) |
| WUT (SATA) | randread, 4k, 65536, 128(0.25GB/Sec) |
| IW (NVMe) | randwrite, 4k, 524288, 8 - 16384 (2GB/Sec) |
| IW (SATA) | randwrite, 4k, 65536, 8 - 16384 (0.25GB/Sec) |

**Table 4: IO depth sensitivity experiment**

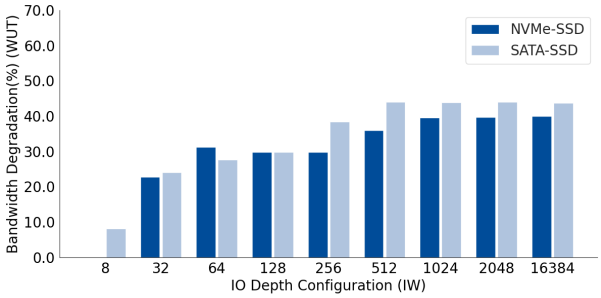


**Figure 5: IO Depth Sensitivity**

## 3.4 IO Type

| **Workload** | **FIO parameters** (IO type, IO size, IO rate, IO depth) |
|---|---|
| WUT(NVMe) | randread, 4k, 524288, 128 (2GB/Sec) |
| WUT (SATA) | randread, 4k, 65536, 128 (0.25GB/Sec) |
| IW (NVMe) | randread, 4k, 524288, 128 (2GB/Sec)<br>randwrite, 4k, 524288, 128 (2GB/Sec) |
| IW (SATA) | randread, 4k, 65536, 128 (0.25GB/Sec)<br>randwrite, 4k, 65536, 128 (0.25GB/Sec) |

**Table 5: IO Type Sensitivity Experiment**

This section analyzes the IO interference impact of read vs. write workloads. So we only vary the IO type of the IWs (Table 5). From Figure 6, we can see that the delta between the bandwidth degradation caused by read and write workloads

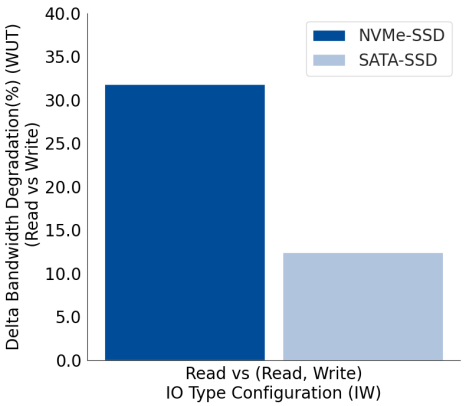


**Figure 6: IO Type Sensitivity**

is ≈ 30% for NVMe SSD and ≈ 10% for SATA SSD. This essentially happens as writes are significantly more expensive than read requests due to the physical nature of SSD discussed in section 2.3.

## 3.5 Spatial IO Access Patterns

| **Workload** | **FIO parameters** (IO type, IO size, IO rate, IO depth) |
|---|---|
| WUT (NVMe) | randread, 4k, 524288, 128 (2GB/Sec)<br>read, 4k, 524288, 128 (2GB/Sec) |
| WUT (SATA) | randread, 4k, 65536, 128(0.25GB/Sec)<br>read, 4k, 65536, 128 (0.25GB/Sec) |
| IW (NVMe) | randwrite, 4k, 524288, 128 (2GB/Sec)<br>write, 4k, 524288, 128 (2GB/Sec) |
| IW (SATA) | randwrite, 4k, 65536, 128(0.25GB/Sec)<br>write, 4k, 65536, 128 (0.25GB/Sec) |

**Table 6: IO Access Pattern Sensitivity Experiment**

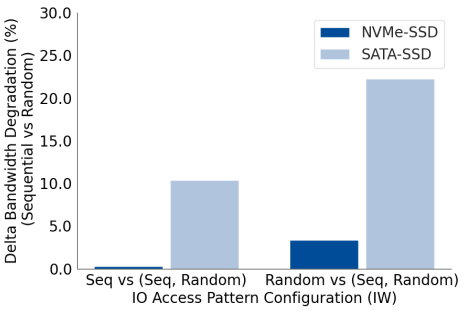


**Figure 7: Spatial IO access pattern Sensitivity**

This section analyzes the interference impact of spatial IO access patterns, such as sequential vs. random. In this experiment, we define two read-only WUTs and two write-only IWs for both NVMe and SATA SSD, only varying their IO access patterns (Table 6). Then we run all four configuration permutations. For NVMe SSD, Figure 7 shows that the delta between bandwidth degradation caused by sequential and random IW is less than ≈ 5%. We attribute this behavior to the write allocation policies described in section 2.1. The probability of accessing a particular flash chip is the same for all chips as long as a sufficient number of outstanding requests and some IO request buffering capability exists in the system [35, 44]. However, for the SATA SSD, we see a higher delta between the performance degradation caused by the random and sequential IWs, ≈ 10% for sequential and ≈ 20% for random WUT. This can be explained by our discussion in section 3.2, where we show that operations with a shorter completion time could maintain a higher IO rate and suffer less IO interference. Sequential operations are more efficient than random operations as SSD manufacturers employ optimization techniques, such as prefetching and IO striping [9, 46, 71], increasing overall IO efficiency. As a result, the more efficient sequential WUTs suffer less. In the SATA SSD, such behavior is more evident due to limited hardware resources and less efficient write operations than NVMe SSD.

## 3.6 IO Offset Alignment

| **Workload** | **FIO parameters** (IO type, IO size, IO rate, IO depth) |
|---|---|
| WUT (NVMe) | randread, 4k, 524288, 128 (2GB/Sec) |
| WUT (SATA) | randread,4k, 65536,16 (0.25GB/Sec) |
| IW (NVMe) | randwrite, 4k, 524288, 128 (2GB/Sec)<br>write:17k, 4k, 524288, 128 (2GB/Sec) |
| IW (SATA) | randwrite, 4k, 65536, 128(0.25GB/Sec)<br>write:17k, 4k, 65536, 128(0.25GB/Sec) |

**Table 7: IO offset Alignment Sensitivity Experiment**

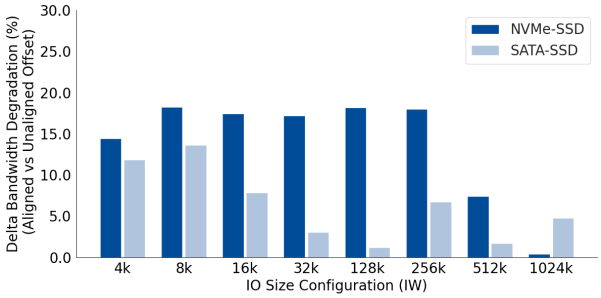


**Figure 8: IO interference due to IO offset alignment**

To explore the IO interference caused by different IO offset alignments, we devise a single WUT and several IWs that

generate two types of IWs. IWs with page-aligned and page-unaligned offset IOs. Note that using FIO with "write:17k" IO type performs page unaligned offset IOs by skipping 17k after each write operation. We do this for multiple IO sizes (Table 7). We only explain the results for SATA SSD; similar observations apply to NVMe SSD. Figure 8 shows that the delta between the bandwidth degradation caused by aligned and unaligned offset IWs is ≈ 10%, and it keeps decreasing as the IO size increases. We attribute this delta to RMW (read-modify–write) operations caused by unaligned offset IOs explained in section 2.2. The decrease in IO interference impact is explained in section 3.1, where an increase in IO size leads to lower unaligned operations than aligned operations. Note that un-aligned offset IOs shows a higher IO interference impact than un-aligned size IOs, as crossing the page boundary can cause RMW operations on two pages, making it significantly expensive. For instance, 4k unaligned IO offset cause two RMW operations as no direct flash writes can be performed. While in the case of 6k IO size on aligned offset cause only one RMW operation.

## 3.7 Temporal IO Access Patterns

| **Workload** | **FIO parameters** (IO type, IO size, IO rate, IO depth, startdelay, thinktime, thinktime) |
|---|---|
| WUT (NVMe) | randread,4k, 524288,32, 0, 0, 0 (Avg:2GB/Sec, Std:0) |
| WUT (SATA) | randread,4k, 65536,32, 0, 0, 0 (Avg:0.25GB/Sec, Std:0) |
| IW Uniform (NVMe) | randwrite, 4k, 262144,128, 0, 0, 0 (Avg:1024MB/s, Std:0) |
| IW Bursty (NVMe) | randwrite, 4k, 53608,128, 1s, 1s, 1s<br>randwrite, 4k, 470680,128,0,1s,1s<br>(Avg: 1024MB/s, Std: 1152MB/s) |
| IW Uniform (SATA) | randwrite, 4k, 32768,128, 0, 0, 0 (Avg:128MB/s, Std:0) |
| IW Bursty (SATA) | randwrite, 4k, 12496,128, 1s, 1s, 1s<br>randwrite, 4k, 50000,128,0,1s,1s<br>(Avg: 128MB/s, Std: 144MB/s) |

**Table 8: Uniform vs Bursty IO Sensitivity Experiment**

This section analyzes the degree of interference caused by temporality of IO access patterns. In particular, we compare bursty workloads against uniform workloads. As per our knowledge, no prior work has considered the varying IO interference of temporal access patterns, although bursty workloads are common in real-world scenarios. For this experiment, we define two IWs. The first IW generates a uniform IO rate (uniform IW), i.e., the bandwidth remains

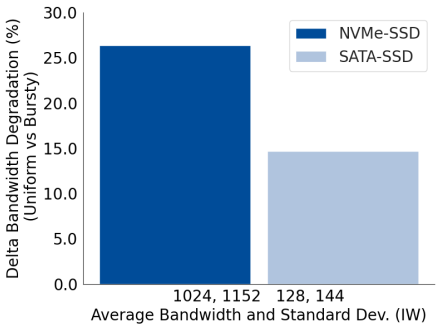


**Figure 9: Temporal (uniform vs. bursty) IO access pattern induced performance degradation**

the same throughout its lifetime, while the other workload (bursty IW) exhibits a variable IO rate. To generate the bursty IW and control the IW's burstiness, we define two FIO jobs that generate varying IOPS during their execution. We utilize FIO job parameters such as *startdelay*, *thinktime* and *thinktime_iotime* (the last three parameters in Table 8). To setup a single bursty IW, we synchronize four FIO jobs by overlapping their start delay (*startdelay*), idle phase (*thinktime*), and IO phase (*thinktime_iotime*), generating an average bandwidth, standard deviation values of approximately 1024, 1152, and 128, 144 for NVMe-SSD and SATA-SSD experiment respectively.

We explain the results for NVMe-SSD; similar observations apply to the SATA-SSD experiment. Figure 9 shows that the delta between the bandwidth degradation caused by uniform vs. bursty workload is approximately ≈ 25%, which we explain as follows. The aggregated IO bandwidth of the WUT (2GB/s) and uniform IW (1 GB/s) is accommodated by the peak bandwidth of the device (3 GB/s), as shown in Table 1. As long as the aggregate bandwidth is lower than or equal to the peak bandwidth of the device, performance interference is limited. However, for bursty interference workloads, the real-time bandwidth varies over time and sometimes exceeds the device's peak bandwidth, causing interference. As a result, burstiness affects the available percentage of peak bandwidth that can be utilized without causing interference showing the insufficiency of average IO rate and bandwidth to predict performance interference. Next, we discuss our proposed approach to represent burstiness.

**Representing Burstiness:** The burstiness of a workload can be described with a time series of IOPS values. However, such time series have an unbounded size and are difficult to aggregate. Furthermore, we would need to know the exact phase difference between the WUT and the IW to compute interference. We propose a different technique to represent the burstiness of a workload based on the standard deviation from its mean. To evaluate the applicability of the standard

deviation, we devise an experiment where we expose the WUT to several bursty IWs with variable IO rates. Our results in Figure 10 demonstrate a strong correlation between the average bandwidth and standard deviation of the IWs with the average bandwidth achieved by WUT, showing that standard deviation is a good proxy for bursty interference.

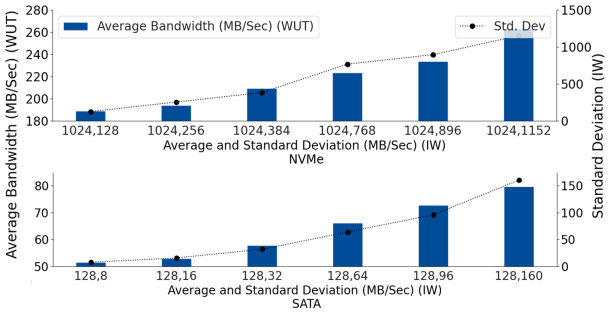


**Figure 10: Correlating Std. Dev. and IO Interference**

## 4 GRAY-BOX FEATURE APPROACH

Prior works [6, 7, 12, 13, 15, 52] lack important features while building IO Interference prediction models. Consequently, they suffer from the following shortcomings. First, they cannot represent IO interference caused by SSD internals, which is important because IO interference changes significantly corresponding to **IO size and offset alignment** due to the physical nature of the SSD (discussed in section 3.1 and 3.6). Second, traditional features such as average IO bandwidth alone cannot represent the **bursty** behavior of the workloads and fails to represent their IO interference impact, which can be significantly different from uniform workloads (see section 3.7).

To address these issues, we define our Gray-box feature set that considers workload properties corresponding to SSD internals to represent IO interference impact accurately. We refer to our features as Gray-box because we relate each SSD property affecting IO interference (discussed in section 2) while representing the IO workloads.

### 4.1 Gray-box Features

Table 9 lists our proposed features for representing an IW. We utilize 8 features, each considering IO type (read and write) and IO size alignment classes (page aligned and page unaligned size), following our analysis in section 2 and section 3. Since a single IW could have varying IO sizes, we represent the IO sizes used by the IW by taking the weighted mean of all the IO sizes observed in the workload corresponding to their IO rate. To consider the IO Interference impact of burstiness of the workload (discussed in section 3.7), we utilize the average and standard deviation of the IO rate and IO bandwidth, which are valid proxies for representing burstiness (Section 3.7). In our feature set, we utilize both IO rate and IO bandwidth as these features also represent queuing contention inside an SSD. IO bandwidth, together with weighted mean size and avg IO rate, represents contention to utilize resources such as flash chips. To consider the effect of the IO offsets (discussed in section 3.6), we calculate the ratio of unaligned offset IOs in their respective categories. Similarly, we calculate the ratio of random IOs capturing the effect of spatial IO access patterns (discussed in section 3.5). In section 6, we evaluate the prediction accuracy enabled by our proposed features, demonstrating that our proposed features are sufficient to represent IO interference impact accurately. Next, we describe our methodology for feature aggregation from multiple workloads and its importance.

| Features | Read / Write | |
|---|---|---|
| | Aligned Size IOs | Unaligned Size IOs |
| Weighted Mean Size | $WM_{as}$ | $WM_{us}$ |
| Average IO Rate | $AR_{as}$ | $AR_{us}$ |
| Std. Dev. IO Rate | $SR_{as}$ | $SR_{us}$ |
| Average IO BW | $AB_{as}$ | $AB_{us}$ |
| Std. Dev. IO BW | $WM_{as}$ | $WM_{us}$ |
| Unaligned Offset Ratio | $U_{as}$ | $U_{us}$ |
| IO Depth | $D$ | |
| Random Ratio | $R_{ratio}$ | |

**Table 9: Gray-box Features**

### 4.2 Gray-box Feature Aggregation

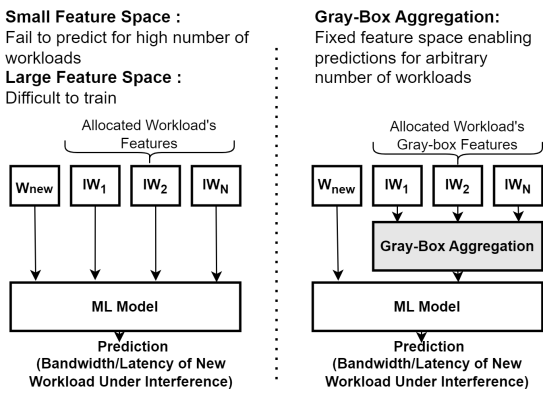


**Figure 11: Supporting Arbitrary Number of Workloads**

In a real-world environment, estimating the number of workloads that can share the SSD is difficult as it is subject to their SLOs. For instance, while a few workloads with strict

SLOs can be colocated, many workloads with flexible SLOs can be colocated on the same SSD. Essentially, the ML model should be able to predict the IO Interference impact of the arbitrary number of workloads. Nevertheless, predicting IO interference with an arbitrary number of workloads poses a challenge as training ML models with a fixed number (N) of workloads can render the model ineffective in predicting IO interference beyond N workloads. Alternatively, increasing the feature space to support predictions for a higher number of workloads makes the training and inference difficult and sometimes infeasible due to the curse of dimensionality (Figure 11) [76]. Therefore, we propose our **Gray-box feature aggregation** technique to aggregate the individual features of N IWs without losing their IO interference characteristics. This technique enables the ML model to generalize its predictions to an arbitrary number of workloads. Next, we explain the process for feature aggregation.

First, we preprocess the block traces of interfering workloads and represent them with our proposed **Gray-box features** (Section 4.1). As shown in Table 9 we first separate IO traffic according to their IO type and size alignment. In the following, we describe how to aggregate features in the Table's first column (Aligned Size IOs). The same methodology applies to calculating the feature values in another column. To calculate the aggregated Weighted Mean Size ($WM_{as}$) of N IWs, we use equation 3 using average rate as weight ($AR_{as}$). To aggregate the average and standard deviation values of IO rate and bandwidth ($AR_{as}$, $SR_{as}$, $AB_{as}$, $SB_{as}$), we utilize equations 4 and 5, respectively. IO depth features should be aggregated by summing up the IO depths of N IWs. For calculating the aggregated ratios, we utilize equation 6, where we first multiply the fraction of unaligned offset ratio by the average IO rate ($AR_{as}$) for all the IWs and then calculate the aggregated ratio(equation 6). Sequential to random ratio is aggregated similarly; the only difference is that the fraction is multiplied by the average IO rate across both aligned and unaligned IO size traffic (average aligned($AR_{as}$) + average unaligned IO rate($AR_{us}$)).

$$\text{Agg\_Weighted\_Mean}(\text{Size}_\text{i}, \text{Rate}_\text{i}) = \frac{\sum_{i=1}^{N}(Size_\text{i} \cdot Rate_\text{i})}{\sum_{i=1}^{N} Size_\text{i}} \tag{3}$$

$$\text{Aggregate\_Avg}(\text{X}_\text{i}) = \sum_{i=1}^{N} X_\text{i} \tag{4}$$

$$\text{Aggregate\_Std}(\text{Y}_\text{i}) = \sqrt{\sum_{i=1}^{N} {Y_\text{i}}^2} \tag{5}$$

$$\text{Aggregated Ratio} = \frac{\sum_{i=1}^{N} U_i \times \text{Avg IO Rate}_i}{\sum_{i=1}^{N} \text{Avg IO Rate}_i} \tag{6}$$

# 5 USAGE MODEL

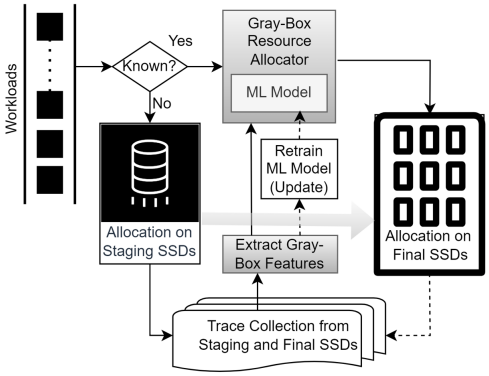


**Figure 12: Usage Model**

**Algorithm 1** Resource Allocation

```
1:  function GRAY_BOX_RESOURCE_ALLOCATION(workload)
2:      GroupCount, allocations ← global_allocations
3:      assigned ← False
4:      slo_violation ← False
5:      for i in range (GroupCount) do
6:          t_group ← allocations[i] + workload
7:          for j in range (len(t_group)) do
8:              w ← t_group[j]
9:              g ← [all the workloads in t_group except w]
10:             if SLO_VIOLATION(w, g) then
11:                 slo_violation ← True
12:                 break
13:             end if
14:         end for
15:         if not slo_violation then
16:             allocations[i].append(workload)
17:             break
18:         end if
19:     end for
20:     if not assigned then
21:         allocations.append([workload)
22:         GroupCount ← GroupCount + 1
23:     end if
24:     global_allocations ← GroupCount, allocations
25: end function
```

Figure 12 shows a high-level usage model of the Gray-box Resource Allocator. Submitted workloads are retrieved from a queue and looked up in a database of previously seen workloads. If the workload is already known (Yes), the Gray-box Allocator queries the SSD cluster for existing allocations to determine candidate SSDs that could absorb the new workload. To handle workload identification, we use a unique

identifier corresponding to extracted gray-box features from their trace profiles. In the real world, it can be a workload ID in a private cloud or a signature in a public cloud. We recognize that signature identification can be difficult and may require more complex mechanisms capturing various phases of a workload [4, 53, 59]. We intend to investigate such mechanisms in our future work.

The allocator utilizes Algorithm 1 to find the best candidate location. In particular, it iterates through all candidate locations with enough storage capacity to house the workload and then utilizes the ML model to assess whether the new workload (WUT) suffers or causes SLO violations when colocated with the preexisting workloads (IW) on that SSD. The two-nested loop in Algorithm 1 is required as in addition to analyzing whether the new workload can be scheduled without violating its SLO, we also need to ensure that the already provisioned workloads' SLOs are not violated. As a result, the worst-case execution time of the allocation mechanism is given by the number of candidate locations (SSDs that provide sufficient storage capacity) times the number of pre-existing workloads on these candidate locations. Each model inference task takes less than 400 microseconds on the Intel Xeon server (see Table 1) we use for our experiments, and hence for a 1000 SSD cluster allocation can be performed in less than 10 seconds. The two-nested loop in Algorithm 1 can be unrolled and parallelized. By batching inference tasks and leveraging GPUs, we envision allocation can be performed in a few milliseconds.

For predicting the expected performance for the new workload (bandwidth or latency) under interference, the ML model utilizes Gray-box features. These features encompass both the new workload's gray-box features and the existing workloads' aggregated gray-box features, serving as input features for the model (Figure. 11). The ML model is trained using the Random Forest (RF) regression algorithm, which has shown to provide high accuracy for IO Interference prediction [15]. We utilize the Scikit-learn [58] ML library with the default parameters and 100 estimators: RandomForestRegressor(n_estimators=100, random_state=42). To handle a variety of SSD landscapes today, we train separate models for each SSD type, incorporating their different IO interference characteristics corresponding to their internals as shown in section 3.

If the new workload is unknown and cannot be found in the database of previously seen workloads, it is assigned to a staging SSD, where workload traces, and performance data are collected. The trace is analyzed to extract Gray-box features, and a new entry is added to the database for future observations of the same workload. At this point, the workload can be migrated, or the SSD can be logically moved to the storage cluster. To ensure that the accuracy of the ML model does not degenerate over time, it can be periodically retrained to accommodate new types of workloads. Trace collection is performed for 10 seconds after the workload reaches a steady state and before features are extracted in the staging area. At this point, the staging SSD can be added to the main SSD cluster. In a real-world environment, workload characteristics can change over their lifetime. To handle such workload drifts, online performance tracking (SLO violations) with cost-benefit analysis can be leveraged to trigger re-profiling and readjusting resource allocation [5, 18, 20, 59, 61]. However, such mechanisms might incur profiling overhead and downtime during data migration. We plan to explore such overheads in the future.

# 6 EVALUATION

We will first describe our evaluation methodology and then evaluate the utilization and TCO improvements provided by our proposed Gray-box technique. To provide additional insights, we then provide a thorough accuracy and feature sensitivity analysis of the ML model used by the Gray-box Allocator.

## 6.1 Experimental Setup and Methodology

**Baselines.** Due to the neglect of the feature aggregation problem in prior literature, no existing studies have put forward a feature aggregation technique specifically designed for SSDs. Therefore we compare our Gray-box technique against the *Weighted-mean* baseline proposed for HDDs by Park et al. [55]. Weighted-mean aggregates interference workloads by computing the simple weighted mean of the IO sizes, IO depths, sequential to random ratios, average IO rates, and average IO bandwidths. The Weighted-mean technique does not consider temporal behavior (burstiness) nor SSD internals (IO offset and size misalignment). Weighted-mean utilizes the same Random Forests (RF) based regression algorithm as Gray-box (with different features). For the TCO and SLO analysis, we also compare against a quota-based mechanism that is commonly used in existing systems. The quota-based mechanism assigns each workload its worst-case quota (overprovisioning rate) required to meet a certain SLO.

**ML Methodology.** To evaluate our Gray-box ML model, we evaluate five performance metrics: Average bandwidth (MB/Sec), Average Latency (microseconds), and Tail latency (90th, 95th, and 99th percentile in microseconds) for two evaluation metrics: R2 score and Mean Absolute Error (MAE). R2 score [68] is the coefficient of determination representing the proportion of feature data that correlates with labels. We use MAE along with R2 score as it does not represent the accuracy of predictions. We train separate models for each performance metric corresponding to its IO type(Read/Write)

and randomly choose 75% of the data for training and 25% for validation.

**Input Data Sets.** We evaluate the accuracy of our approach on real-world block-level IO traces from Alibaba Cloud [42] and Tencent [43] live production servers, obtained via the SNIA [64] and Github [2] open source trace repositories. In particular, these traces are used for the evaluation in sections 6.2, 6.3, and 6.5. To generate the actual workloads, we randomly select 500 ten-second workload samples from these traces. Each sub-trace is viewed as a separate workload, exhibiting its own IO access pattern.

For the experiment in section 6.4, we utilize synthetic traces to precisely control specific workload characteristics such as unaligned accesses. To prepare synthetic traces, we prepare 500 constrained-random synthetic FIO workloads with varying workload properties, run them on the SSD in isolation for 10 sec, and collect their traces.

**Evaluation and Test Set.** To prepare training and evaluation data, we randomly select 2-16 workload traces, run them on the SSD simultaneously and collect the performance observed by each trace-replay (workload), which we use as our labels. We limit ourselves to 16 traces considering the maximum bandwidth supported by the device. Then we calculate Gray-box features for each trace. For each run, we alternatively consider one workload as WUT and the remaining as IWs. Features contain Gray-box features for WUT and aggregated Gray-box features for IWs. Performance numbers of WUT are used as labels. In such a way, we have prepared more than 4000 data points (2000: synthetic 2000: real-world). We randomly choose 75% of the data for training and 25% for validation.

## 6.2 Optimized Gray-box Allocation

Our proposed Gray-box Allocator considers comprehensive workload characteristics and SSD internals to improve interference prediction and, as a result, increases resource utilization while reducing SLO violations. Here we evaluate the number of required SSDs to house 1000 workloads considering different SLO levels.

Figure 13 compares our Gray-box approach against the Weighted-mean and quota-based baselines for different SLOs. If tenants are unwilling to accept a single percentage performance decrease when co-locating other workloads, none of the techniques can improve utilization. As the SLO is relaxed, e.g., if 5% performance degradation is acceptable, Gray-box can pack the 1000 workloads with 327 SSDs, while Weighted-mean requires over 600 SSDs. Providing more accurate interference predictions allows Gray-box to aggressively combine workloads on a shared SSD without violating SLOs. We will further examine this aspect in section 6.3. The quota-based mechanism requires much more relaxed SLO configurations to improve utilization. The key disadvantage of the quota-based mechanisms is that it considers the worst-case interference across all workloads. In contrast, Gray-box and Weighted-mean consider pairwise interference between 2 sets of workloads. For an SLO of 10%, Gray-box provides 31% higher resource utilization over Weighted-mean and 60% over quota-based allocation. Similarly, for the same utilization rate(<700 SSDs), while Weighted-mean and Quota-based allocations only enforce an SLO of 5%, Gray-box can enforce a stricter SLO of 2%. In summary, Gray-box improves resource utilization by up to 60% when considering an SLO of 5%, or it can enforce an up to 2.5× stricter SLO over Weighted-mean when utilizing up to 68% of the available SSDs.

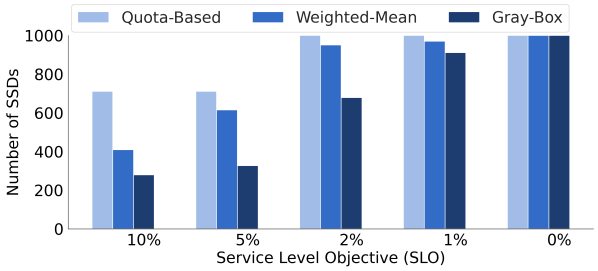


**Figure 13: Resource Allocation Comparison**

## 6.3 Resource Utilization and SLO Compliance

In the previous section, we showed how Gray-box can improve resource utilization. To provide additional insight into how Gray-box outperforms Weighted-mean, we pick 30 random samples from our prediction results and measure the prediction error corresponding to the actual bandwidth. A high positive error means that the ML model underpredicted the interference, leading to an SLO violation, whereas a high negative error means the ML model overpredicted the interference, causing low-resource utilization. This experiment shows the quality of the ML models, in particular, how well they can approximate the actual Interference. Figure 14 represents the prediction error (in %) of Gray-box and Weighted-mean, corresponding to the actual bandwidth observed by the WUT in the presence of interference. As can be seen, for a chosen SLO of 10%, Weighted-mean shows 14 (out of 30) violations (e.g., workload mixes 26, 27), whereas Gray-box suffers from none. Furthermore, in nine cases, the Weighted-mean technique falls below the 10% resource utilization threshold reducing efficiency and causing high TCO. For instance, for workload mixes 12 and 13, the Weighted-mean technique predicts the bandwidth ≈ 15 and 25% lower than the actual bandwidth preventing co-locating them with their respective workload mixes.

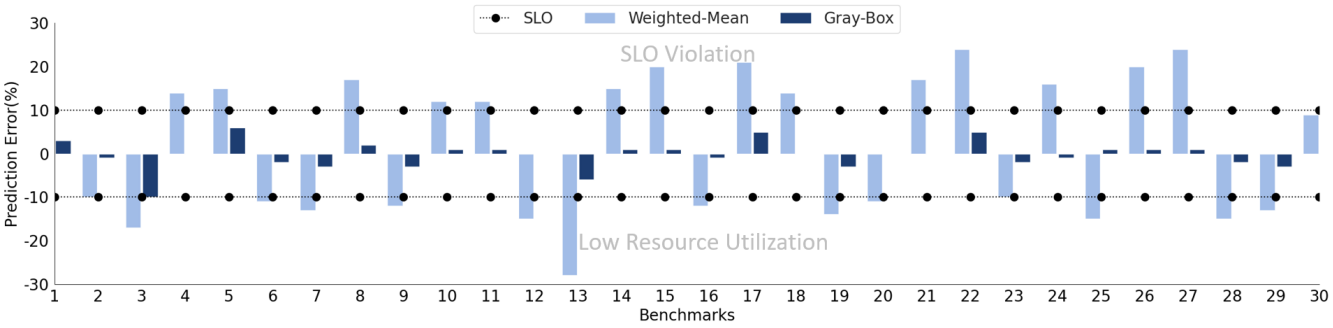


**Figure 14: Effectiveness of Gray-box for preventing SLO violations and increasing Resource utilization**

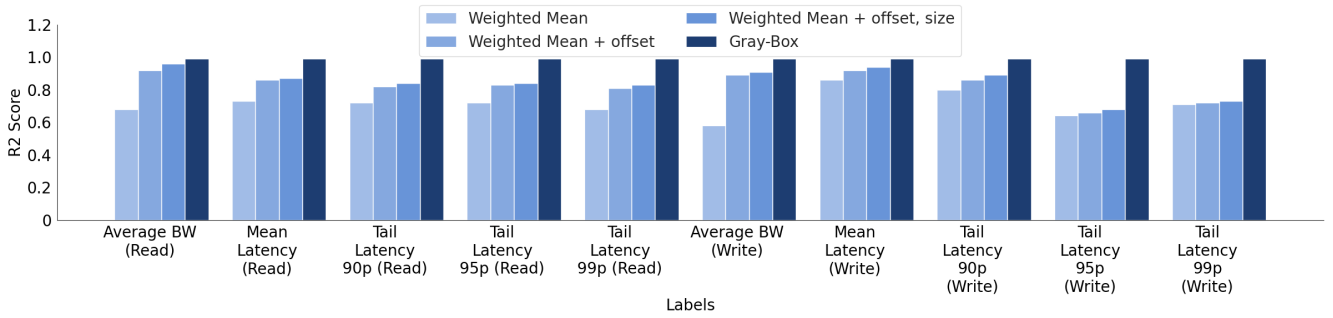


**Figure 15: Gray-box ML Feature Sensitivity: Synthetic Data (R2 score)**

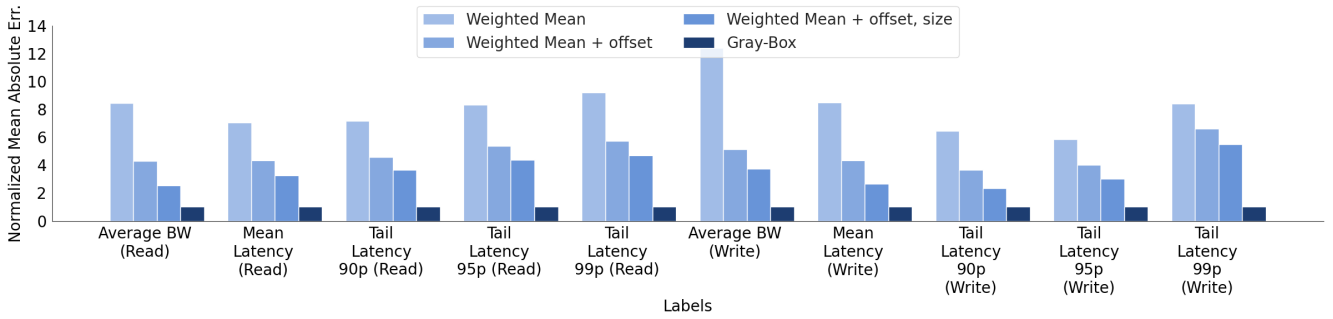


**Figure 16: Gray-box ML Feature Sensitivity: Synthetic Data (MAE)**

In summary, Gray-box always shows higher resource utilization than Weighted-mean while maintaining SLO compliance, as its prediction closely follows the actual performance.

## 6.4 ML Features Sensitivity Study

Our proposed Gray-box technique considers SSD internals and additional workload characteristics such as burstiness to improve prediction accuracy over the Weighted-mean baseline. In the following, we evaluate the impact of these individual features on prediction performance. In particular, we evaluate the effect of considering IO offsets, IO sizes, and burstiness by adding the features one by one (Gray-box has all features enabled). Figure 15 shows the R2 score for predicting the WUT's performance when running against an IW. The x-axis shows different performance metrics, such as latency and bandwidth and the y-axis shows the R2 score. We train (10 x 4) ML models, for each performance metric to evaluate how each of the newly proposed features contributes to accuracy.

To evaluate the impact of the features on the actual prediction, we calculate the Normalized Mean Absolute Error (NMAE), considering predictions with Gray-box features

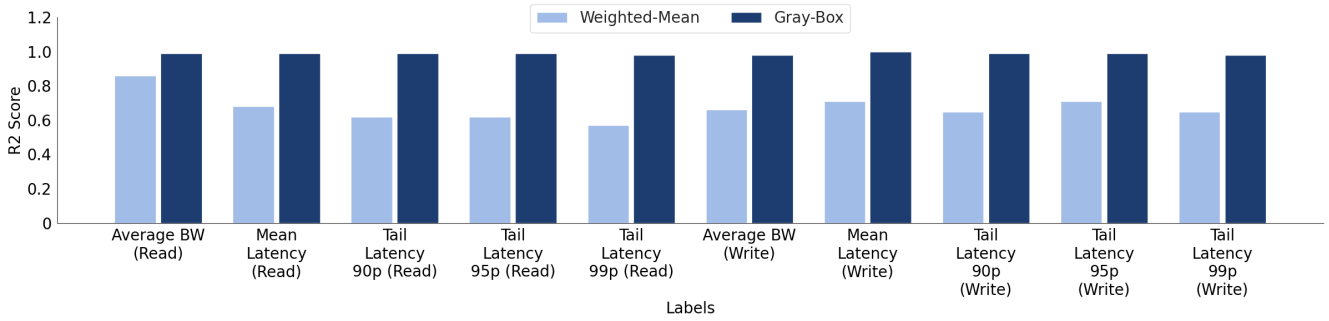


**Figure 17: Weighted-Mean Vs. Gray-box Prediction Accuracy: Real-World Data (R2 score)**

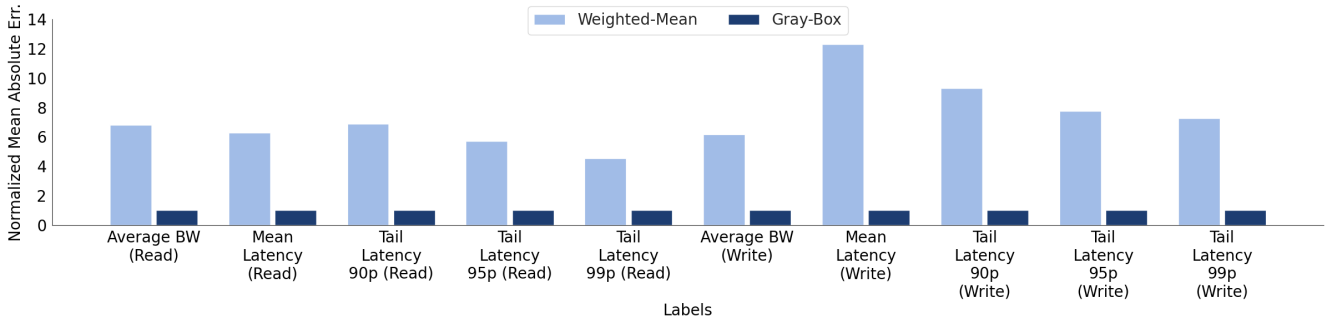


**Figure 18: Weighted-Mean Vs. Gray-box Prediction Accuracy: Real-World Data (MAE)**

as a baseline (shown in Figure 16). We can see how adding our proposed Gray-box features considering burstiness and SSD Internals (offset, size alignment) improves the MAE. In particular, NMAE for the Weighted-mean prediction is 6 to 14 times higher than the Gray-box prediction for all the performance metrics. While Weighted-mean + offset features improve the prediction accuracy, their NMAE is still 4 to 8× higher, which reduces to 2 to 6× after adding size alignment features. This shows how adding our proposed features decreases the MAE and improves the IO Interference's prediction accuracy.

## 6.5 Prediction Accuracy: Real World data

Figure 17 shows the R2 score to predict the performance of real-world WUT when run against IW. The interpretation of the graph is analogous to Figure 15. Similar to synthetic data, we could see that the R2 score for the Weighted-mean is significantly lower than the Gray-box features. For all the performance metrics, while R2 score for Weighted-mean remains lower than 0.87, for the Gray-box, it is more than 0.98. Fundamentally, this happens as the Weighted-mean features disregard the non-uniform IO interference impact of workload properties such as burstiness and SSD internals (IO size and offset alignment), making IO interference impact difficult to learn. Similarly, Figure 18 shows Normalized Mean Absolute Error analogous to Figure 16. For each performance metric, MAE for the Weighted-mean is 4 to 12× higher than the Gray-box prediction.

In summary, we show that our proposed Gray-box technique enables accurate IO interference prediction essentially due to first accurately representing the IO interference impact of SSD internals and non-uniform workload characteristics (Section 4.1) and second aggregating features from multiple IWs (Section 4.2) without losing their IO interference characteristics.

# 7 RELATED WORK

Prior research [12, 13, 15, 21, 40, 52] has proposed several approaches to address the problem of IO interference in HDD based storage technologies. As the internal architecture of HDDs differs substantially from SSDs, these techniques are not applicable for our work. In particular, these suffer reduced accuracy as their considered feature set does not consider non-uniform workload characteristics and SSD internals. For instance, Noorshams et al. [52] use average IO throughput, IO size, threads, file set size, access pattern (random, sequential), and read ratio; while Chiang et. al [13] only use CPU utilization and read/write requests per second.

We have shown that these features alone do not represent accurate IO interference, as they do not express non-uniform nature workloads as discussed in section 3.1 and 3.7. Furthermore, these prior works do not use any workload aggregation technique limiting their predictions to a fixed number of workloads. For instance, Dartois et al. [15] use ML techniques to model SSD IO performance in the presence of up to five interfering workloads. Their approach also fails to predict tail latency interference.

Bhimani [6] proposes batching containerized workloads to improve overall resource utilization and fairness based on a few proposed guidelines. Although the guidelines help improve overall resource utilization, they are insufficient to efficiently eliminate IO interference. Chiang [12] proposes a contention-aware placement strategy based on ML-based clustering algorithms for container placement. Their work focuses on improving overall resource utilization, while ignoring performance interference. In addition, their placement strategy is based on CPU utilization, memory utilization, and IO utilization, ignoring SSD specific properties. Gulati proposes Basil [19] and Pesto [20] to enable load balancing of IO workloads. Those systems use the LQ-slope performance model representing the latency-to-queue depth ratio showing a linear relationship between latency and queue depth. While this model can predicts overall performance degradation it cannot determine the interference impact on individual workloads. Kim [34] designed classification-based ML models to predict SSDs performance saturation using kernel IO statistics and workload features. However, those features do not represent the interference impact of the non-uniform IW and SSD internals on the workloads. Queuing models [40] have also been used to predict IO interference impact on the performance of workloads. However, first of all, they base their work on HDDs. Second, the feature set they use does not represent the IO interference impact of spatial IO access patterns such as unaligned offset IOs, which leads to significant IO interference in the case of SSDs as discussed in section 3.6.

Apart from performance modeling, researchers proposed performance isolation techniques that require changes to the internal SSD resources [11, 27, 29, 30, 32, 41, 47, 62, 65, 67]. For instance, they propose static or dynamic partitioning techniques for allocating SSD resources to workloads. While these techniques can reduce interference, they require time-consuming manual application configuration, often wasting resources due to static partitioning as they can only support a small number of workloads or tenants. A dynamic resource allocation scheme can help this situation. However, resource allocation can be skewed as they lack workload context, reducing overall resource utilization. Kim [35] introduced VA-LVM, a logical volume manager, creating logical volume mapping to SSD's internal volumes. While VA-LVM provides better performance isolation, it is limited by the availability of SSD's internal volume. In contrast, our work focus on sharing overall SSD that can improve resource utilization while preventing SLO violations.

Another category of works proposed techniques to throttle IO requests based on various feedback-based heuristics, leveraging system statistics [17, 21, 45, 51, 54, 57, 73]. These approaches lack an understanding of the underlying hardware and usually only react to an observed performance degradation while our approach can prevent such performance degradation in the first place. Prior works [39] utilize execution time degradation for IO scheduling, however, it requires extensive profiling to generate a model.

Several other works which are mostly orthogonal to ours have proposed machine learning models for improving SSD performance to improve performance [9, 60] , increase lifetime [8, 28], and to reduce garbage collection overheads [10].

## 8 CONCLUSION

In this work, we propose and evaluate a novel Gray-box feature representation and aggregation technique to accurately represent IO interference in SSDs. We perform a detailed analysis of IO interference in SSDs, showing how SSD internals affect the IO interference (Section 2). Further, we explore the relationship between different non-uniform workloads and internal SSD architecture characteristics with exhaustive experimentation (Section 3). Then, based on our observations, we define a set of Gray-box features that accurately represent the IO interference impact of SSD Internals. We propose new features not considered by prior work, such as IO size alignment, IO offset alignment, and workload burstiness. To support predictions of an arbitrary number of workloads, we propose a new Gray-box feature aggregation technique to aggregate their features without losing IO interference accuracy. Our evaluation shows that our Gray-box approach outperforms prior works by increasing inference prediction accuracy for latency and bandwidth while reducing the mean absolute error by up to 12×. Furthermore, in contrast to prior work, we show that predicting the IO interference impact on tail latency is indeed feasible. We demonstrate how the Gray-box technique increases resource utilization by up to 60% or enforces up to 2.5× stricter SLOs when provisioning SSD among multiple tenants over prior works.

## 9 ACKNOWLEDGMENTS

We thank the anonymous reviewers and our shepherd, Brian Kroth, for their helpful feedback. This work was generously supported by Samsung and NSF grants CCF-1942754 and CNS-1841545.

# REFERENCES


[1] Nitin Agrawal, Vijayan Prabhakaran, Ted Wobber, John D Davis, Mark Manasse, and Rina Panigrahy. Design tradeoffs for {SSD} performance. In *2008 USENIX Annual Technical Conference (USENIX ATC 08)*, 2008.

[2] Alibaba Group. block-traces. https://github.com/alibaba/block-traces, Accessed 2023.

[3] Jens Axboe. Fio-flexible i/o tester synthetic benchmark. *URL https://github. com/axboe/fio*, 2005.

[4] Jayanta Basak, Kushal Wadhwani, and Kaladhar Voruganti. Storage workload identification. *ACM Transactions on Storage (TOS)*, 12(3):1–30, 2016.

[5] Romil Bhardwaj, Kirthevasan Kandasamy, Asim Biswal, Wenshuo Guo, Benjamin Hindman, Joseph Gonzalez, Michael Jordan, and Ion Stoica. Cilantro:{Performance-Aware} resource allocation for general objectives via online feedback. In *17th USENIX Symposium on Operating Systems Design and Implementation (OSDI 23)*, pages 623–643, 2023.

[6] Janki Bhimani, Zhengyu Yang, Ningfang Mi, Jingpei Yang, Qiumin Xu, Manu Awasthi, Rajinikanth Pandurangan, and Vijay Balakrishnan. Docker container scheduler for i/o intensive applications running on nvme ssds. *IEEE Transactions on Multi-Scale Computing Systems*, 4(3):313–326, 2018.

[7] Giuliano Casale, Stephan Kraft, and Diwakar Krishnamurthy. A model of storage i/o performance interference in virtualized systems. In *2011 31st International Conference on Distributed Computing Systems Workshops*, pages 34–39. IEEE, 2011.

[8] Chandranil Chakraborttii and Heiner Litz. Improving the accuracy, adaptability, and interpretability of ssd failure prediction models. In *Proceedings of the 11th ACM Symposium on Cloud Computing*, pages 120–133, 2020.

[9] Chandranil Chakraborttii and Heiner Litz. Learning i/o access patterns to improve prefetching in ssds. In *Joint European Conference on Machine Learning and Knowledge Discovery in Databases*, pages 427–443. Springer, 2020.

[10] Chandranil Chakraborttii and Heiner Litz. Reducing write amplification in flash by death-time prediction of logical block addresses. In *Proceedings of the 14th ACM International Conference on Systems and Storage*, pages 1–12, 2021.

[11] Da-Wei Chang, Hsin-Hung Chen, and Wei-Jian Su. Vssd: performance isolation in a solid-state drive. *ACM Transactions on Design Automation of Electronic Systems (TODAES)*, 20(4):1–33, 2015.

[12] Ron C Chiang. Contention-aware container placement strategy for docker swarm with machine learning based clustering algorithms. *Cluster Computing*, pages 1–11, 2020.

[13] Ron C Chiang and H Howie Huang. Tracon: Interference-aware scheduling for data-intensive applications in virtualized environments. In *Proceedings of 2011 International Conference for High Performance Computing, Networking, Storage and Analysis*, pages 1–12, 2011.

[14] Transaction Processing Performance Council. Tpc benchmark h (decision support) standard specification revision 3.0.1. 22.

[15] Jean-Emile Dartois, Jalil Boukhobza, Anas Knefati, and Olivier Barais. Investigating machine learning algorithms for modeling ssd i/o performance for container-based virtualization. *IEEE transactions on cloud computing*, 9(3):1103–1116, 2019.

[16] Diego Didona, Jonas Pfefferle, Nikolas Ioannou, Bernard Metzler, and Animesh Trivedi. Understanding modern storage apis: a systematic study of libaio, spdk, and io_uring. In *Proceedings of the 15th ACM International Conference on Systems and Storage*, pages 120–127, 2022.

[17] Congming Gao, Liang Shi, Mengying Zhao, Chun Jason Xue, Kaijie Wu, and Edwin H-M Sha. Exploiting parallelism in i/o scheduling for access conflict minimization in flash-based solid state drives. In *2014 30th Symposium on Mass Storage Systems and Technologies (MSST)*, pages 1–11. IEEE, 2014.

[18] Google Cloud. Adopting SLOs. https://cloud.google.com/architecture/framework/reliability/adopting-slos, 2021.

[19] Ajay Gulati, Chethan Kumar, Irfan Ahmad, and Karan Kumar. Basil: Automated io load balancing across storage devices. In *Fast*, volume 10, pages 13–13, 2010.

[20] Ajay Gulati, Ganesha Shanmuganathan, Irfan Ahmad, Carl Waldspurger, and Mustafa Uysal. Pesto: online storage performance management in virtualized datacenters. In *Proceedings of the 2nd ACM Symposium on Cloud Computing*, pages 1–14, 2011.

[21] 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, 2020.

[22] Xiao-Yu Hu, Evangelos Eleftheriou, Robert Haas, Ilias Iliadis, and Roman Pletka. Write amplification analysis in flash-based solid state drives. In *Proceedings of SYSTOR 2009: The Israeli Experimental Systems Conference*, pages 1–9, 2009.

[23] Yang Hu, Hong Jiang, Dan Feng, Lei Tian, Hao Luo, and Chao Ren. Exploring and exploiting the multilevel parallelism inside ssds for improved performance and endurance. *IEEE Transactions on Computers*, 62(6):1141–1155, 2012.

[24] Yang Hu, Hong Jiang, Dan Feng, Lei Tian, Hao Luo, and Shuping Zhang. Performance impact and interplay of ssd parallelism through advanced commands, allocation strategy and data granularity. In *Proceedings of the international conference on Supercomputing*, pages 96–107, 2011.

[25] Myoungsoo Jung, Wonil Choi, Shekhar Srikantaiah, Joonhyuk Yoo, and Mahmut T Kandemir. Hios: A host interface i/o scheduler for solid state disks. *ACM SIGARCH Computer Architecture News*, 42(3):289–300, 2014.

[26] Myoungsoo Jung and Mahmut T Kandemir. Sprinkler: Maximizing resource utilization in many-chip solid state disks. In *2014 IEEE 20th International Symposium on High Performance Computer Architecture (HPCA)*, pages 524–535. IEEE, 2014.

[27] Jeong-Uk Kang, Jeeseok Hyun, Hyunjoo Maeng, and Sangyeun Cho. The multi-streamed {Solid-State} drive. In *6th USENIX Workshop on Hot Topics in Storage and File Systems (HotStorage 14)*, 2014.

[28] Saeed Kargar, Heiner Litz, and Faisal Nawab. Predict and write: Using k-means clustering to extend the lifetime of nvm storage. In *2021 IEEE 37th International Conference on Data Engineering (ICDE)*, pages 768–779. IEEE, 2021.

[29] Bryan S Kim. Utilitarian performance isolation in shared {SSDs}. In *10th USENIX Workshop on Hot Topics in Storage and File Systems (HotStorage 18)*, 2018.

[30] Bryan S Kim, Hyun Suk Yang, and Sang Lyul Min. {AutoSSD}: an autonomic {SSD} architecture. In *2018 USENIX Annual Technical Conference (USENIX ATC 18)*, pages 677–690, 2018.

[31] Jae-Hong Kim, Dawoon Jung, Jin-Soo Kim, and Jaehyuk Huh. A methodology for extracting performance parameters in solid state disks (ssds). In *2009 IEEE International Symposium on Modeling, Analysis & Simulation of Computer and Telecommunication Systems*, pages 1–10. IEEE, 2009.

[32] Jaeho Kim, Donghee Lee, and Sam H Noh. Towards {SLO} complying {SSDs} through {OPS} isolation. In *13th USENIX Conference on File and Storage Technologies (FAST 15)*, pages 183–189, 2015.

[33] Jaehong Kim, Sangwon Seo, Dawoon Jung, Jin-Soo Kim, and Jaehyuk Huh. Parameter-aware i/o management for solid state disks (ssds). *IEEE Transactions on Computers*, 61(5):636–649, 2011.

[34] Jaehyung Kim, Jinuk Park, and Sanghyun Park. Machine learning based performance modeling of flash ssds. In *Proceedings of the 2017 ACM on Conference on Information and Knowledge Management*, CIKM '17,

page 2135–2138, New York, NY, USA, 2017. Association for Computing Machinery.

[35] Joonsung Kim, Kanghyun Choi, Wonsik Lee, and Jangwoo Kim. Performance modeling and practical use cases for black-box ssds. *ACM Transactions on Storage (TOS)*, 17(2):1–38, 2021.

[36] Kyusik Kim, Eunji Lee, and Taeseok Kim. Hmb-ssd: Framework for efficient exploiting of the host memory buffer in the nvme ssd. *IEEE Access*, 7:150403–150411, 2019.

[37] Ana Klimovic, Heiner Litz, and Christos Kozyrakis. Reflex: Remote flash = local flash. *ACM SIGARCH Computer Architecture News*, 45(1):345–359, 2017.

[38] Ana Klimovic, Heiner Litz, and Christos Kozyrakis. Selecta: Heterogeneous cloud storage configuration for data analytics. In *2018 USENIX Annual Technical Conference (USENIX ATC 18)*, pages 759–773, 2018.

[39] Anthony Kougkas, Hariharan Devarajan, Xian-He Sun, and Jay Lofstead. Harmonia: An interference-aware dynamic i/o scheduler for shared non-volatile burst buffers. In *2018 IEEE International Conference on Cluster Computing (CLUSTER)*, pages 290–301, 2018.

[40] Stephan Kraft, Giuliano Casale, Diwakar Krishnamurthy, Des Greer, and Peter Kilpatrick. Performance models of storage contention in cloud environments. *Software & Systems Modeling*, 12(4):681–704, 2013.

[41] Miryeong Kwon, Donghyun Gouk, Changrim Lee, Byounggeun Kim, Jooyoung Hwang, and Myoungsoo Jung. {DC-Store}: Eliminating noisy neighbor containers using deterministic {I/O} performance and resource isolation. In *18th USENIX Conference on File and Storage Technologies (FAST 20)*, pages 183–191, 2020.

[42] Jinhong Li, Qiuping Wang, Patrick PC Lee, and Chao Shi. An in-depth analysis of cloud block storage workloads in large-scale production. In *2020 IEEE International Symposium on Workload Characterization (IISWC)*, pages 37–47. IEEE, 2020.

[43] Jinhong Li, Qiuping Wang, Patrick PC Lee, and Chao Shi. An in-depth comparative analysis of cloud block storage workloads: Findings and implications. *arXiv preprint arXiv:2203.10766*, 2022.

[44] Nanqinqin Li, Mingzhe Hao, Huaicheng Li, Xing Lin, Tim Emami, and Haryadi S Gunawi. Fantastic ssd internals and how to learn and use them. In *Proceedings of the 15th ACM International Conference on Systems and Storage*, pages 72–84, 2022.

[45] Ning Li, Hong Jiang, Dan Feng, and Zhan Shi. Storage sharing optimization under constraints of slo compliance and performance variability. *IEEE Transactions on Services Computing*, 12(1):58–72, 2016.

[46] Heiner Litz, Javier Gonzalez, Ana Klimovic, and Christos Kozyrakis. Rail: Predictable, low tail latency for nvme flash. *ACM Transactions on Storage (TOS)*, 18(1):1–21, 2022.

[47] Renping Liu, Xianzhang Chen, Yujuan Tan, Runyu Zhang, Liang Liang, and Duo Liu. Ssdkeeper: Self-adapting channel allocation to improve the performance of ssd devices. In *2020 IEEE International Parallel and Distributed Processing Symposium (IPDPS)*, pages 966–975. IEEE, 2020.

[48] Yi Liu, Shouqian Shi, Minghao Xie, Heiner Litz, and Chen Qian. Smash: Flexible, fast, and resource-efficient placement and lookup of distributed storage. *Proceedings of the ACM on Measurement and Analysis of Computing Systems*, 7(2):1–22, 2023.

[49] Chihiro Matsui, Tomoaki Yamada, Yusuke Sugiyama, Yusuke Yamaga, and Ken Takeuchi. Optimal memory configuration analysis in tri-hybrid solid-state drives with storage class memory and multi-level cell/triple-level cell nand flash memory. *Japanese Journal of Applied Physics*, 56(4S):04CE02, 2017.

[50] Rino Micheloni, Luca Crippa, Cristian Zambelli, and Piero Olivo. Architectural and integration options for 3d nand flash memories. *Computers*, 6(3):27, 2017.

[51] Till Miemietz, Hannes Weisbach, Michael Roitzsch, and Hermann Härtig. K2: Work-constraining scheduling of nvme-attached storage. In *2019 IEEE Real-Time Systems Symposium (RTSS)*, pages 56–68. IEEE, 2019.

[52] Qais Noorshams, Axel Busch, Andreas Rentschler, Dominik Bruhn, Samuel Kounev, Petr Tuma, and Ralf Reussner. Automated modeling of i/o performance and interference effects in virtualized storage systems. In *2014 IEEE 34th International Conference on Distributed Computing Systems Workshops (ICDCSW)*, pages 88–93. IEEE, 2014.

[53] Lu Pang and Krishna Kant. Server-side workload identification for hpc i/o requests. In *Proceedings of the 2nd Workshop on Performance EngineeRing, Modelling, Analysis, and VisualizatiOn Strategy*, pages 15–22, 2022.

[54] Hyunchan Park, Seehwan Yoo, Cheol-Ho Hong, and Chuck Yoo. Storage sla guarantee with novel ssd i/o scheduler in virtualized data centers. *IEEE Transactions on Parallel and Distributed Systems*, 27(8):2422–2434, 2015.

[55] Nohhyun Park, Irfan Ahmad, and David J Lilja. Romano: autonomous storage management using performance prediction in multi-tenant datacenters. In *Proceedings of the Third ACM Symposium on Cloud Computing*, pages 1–14, 2012.

[56] Seon-yeong Park, Euiseong Seo, Ji-Yong Shin, Seungryoul Maeng, and Joonwon Lee. Exploiting internal parallelism of flash-based ssds. *IEEE Computer Architecture Letters*, 9(1):9–12, 2010.

[57] 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'12, page 13, USA, 2012. USENIX Association.

[58] Fabian Pedregosa, Gaël Varoquaux, Alexandre Gramfort, Vincent Michel, Bertrand Thirion, Olivier Grisel, Mathieu Blondel, Peter Prettenhofer, Ron Weiss, Vincent Dubourg, et al. Scikit-learn: Machine learning in python. *Journal of machine learning research*, 12(Oct):2825–2830, 2011.

[59] Pankaj Pipada, Achintya Kundu, Kanchi Gopinath, Chiranjib Bhattacharyya, Sai Susarla, and PC Nagesh. Loadiq: Learning to identify workload phases from a live storage trace. *HotStorage*, 12, 2012.

[60] Devashish Purandare, Pete Wilcox, Heiner Litz, and Shel Finkelstein. Append is near: Log-based data management on zns ssds. In *12th Annual Conference on Innovative Data Systems Research (CIDR'22)*., 2022.

[61] Babak Ravandi, Ioannis Papapanagiotou, and Baijian Yang. A black-box self-learning scheduler for cloud block storage systems. In *2016 IEEE 9th International Conference on Cloud Computing (CLOUD)*, pages 820–825. IEEE, 2016.

[62] Eunhee Rho, Kanchan Joshi, Seung-Uk Shin, Nitesh Jagadeesh Shetty, Jooyoung Hwang, Sangyeun Cho, Daniel DG Lee, and Jaeheon Jeong. {FStream}: Managing flash streams in the file system. In *16th USENIX Conference on File and Storage Technologies (FAST 18)*, pages 257–264, 2018.

[63] I Shin. Improving internal parallelism of solid state drives with selective multi-plane operation. *Electronics Letters*, 54(2):64–66, 2018.

[64] SNIA. Block io traces. http://iotta.snia.org/tracetypes/3, Dec 2001.

[65] Xiang Song, Jian Yang, and Haibo Chen. Architecting flash-based solid-state drive for high-performance i/o virtualization. *IEEE Computer Architecture Letters*, 13(2):61–64, 2013.

[66] Arash Tavakkol, Juan Gómez-Luna, Mohammad Sadrosadati, Saugata Ghose, and Onur 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)*, pages 49–66, 2018.

[67] Shivani Tripathy, Debiprasanna Sahoo, Manoranjan Satpathy, and Madhu Mutyam. Fuzzy fairness controller for nvme ssds. In *Proceedings of the 34th ACM International Conference on Supercomputing*, pages 1–12, 2020.

[68] Michael R Veall and Klaus F Zimmermann. Pseudo-r2 measures for some common limited dependent variable models. *Journal of Economic*

*surveys*, 10(3):241–259, 1996.

[69] Suzhen Wu, Weiwei Zhang, Bo Mao, and Hong Jiang. Hotr: Alleviating read/write interference with hot read data replication for flash storage. In *2019 Design, Automation & Test in Europe Conference & Exhibition (DATE)*, pages 1367–1372. IEEE, 2019.

[70] Minghao Xie and Chen Qian. Reflex4arm: Supporting 100gbe flash storage disaggregation on arm soc. In *OCP Future Technology Symposium*, 2020.

[71] Rui Xu, Xi Jin, Linfeng Tao, Shuaizhi Guo, Zikun Xiang, and Teng Tian. An efficient resource-optimized learning prefetcher for solid state drives. In *2018 Design, Automation & Test in Europe Conference & Exhibition (DATE)*, pages 273–276. IEEE, 2018.

[72] Gala Yadgar, MOSHE Gabel, Shehbaz Jaffer, and Bianca Schroeder. Ssd-based workload characteristics and their performance implications. *ACM Trans. Storage*, 17(1), jan 2021.

[73] Suli Yang, Tyler Harter, Nishant Agrawal, Salini Selvaraj Kowsalya, Anand Krishnamurthy, Samer Al-Kiswany, Rini T. Kaushik, Andrea C. Arpaci-Dusseau, and Remzi H. Arpaci-Dusseau. Split-level i/o scheduling. SOSP '15, page 474–489, New York, NY, USA, 2015. Association for Computing Machinery.

[74] Jung H Yoon and Gary A Tressler. Advanced flash technology status, scaling trends & implications to enterprise ssd technology enablement. *Flash Memory Summit*, 3(3.1):4, 2012.

[75] Yu Zhang, Ping Huang, Ke Zhou, Hua Wang, Jianying Hu, Yongguang Ji, and Bin Cheng. *OSCA: An Online-Model Based Cache Allocation Scheme in Cloud Block Storage Systems*. USENIX Association, USA, 2020.

[76] Arthur Zimek, Erich Schubert, and Hans-Peter Kriegel. A survey on unsupervised outlier detection in high-dimensional numerical data. *Statistical Analysis and Data Mining: The ASA Data Science Journal*, 5(5):363–387, 2012.