Utilitarian Performance Isolation in Shared SSDs

Bryan S. Kim
Seoul National University

Abstract

This paper proposes a utilitarian performance isolation (UPI) scheme for shared SSD settings. UPI exploits SSD's abundant parallelism to maximize the utility of all tenants while providing performance isolation. Our approach is in contrast to static resource partitioning techniques that bind parallelism, isolation, and capacity altogether. We demonstrate that our proposed scheme reduces the 99th percentile response time by 38.5% for a latency-critical workload, and the average response time by 16.1% for a high-throughput workload compared to the static approaches.

1 Introduction

Thanks to its low latency, collectively massively parallelism, and high density, SSDs are now critical components in today's large storage systems from cloud services to high-performance computing environments. However, SSDs exhibit substantial performance instabilities and variations due to background management tasks [6, 13, 15, 21]. These problems are further exacerbated in shared storages, where multiple tenants interfere with each other [8, 11, 12, 20], causing a so-called *noisy neighbor* effect.

To curtail the effects of inter-tenant I/O interferences, the current trend is to configure the SSD into multiple isolated regions [3, 10, 20]. By partitioning it internally and assigning each tenant a separate region, either at the channel-level or chip-level, each would be physically isolated and the workload of one tenant would not degrade the performance of another. However, statically partitioning internal resources can be detrimental across multiple dimensions. First, only a fraction of the physical storage space can be used for each tenant. Second, the average performance suffers by limiting the overall parallelism. Lastly, the quality of service (QoS) degrades on highly contended flash resources as the load cannot be effectively distributed.

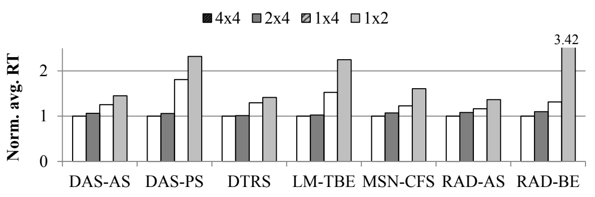
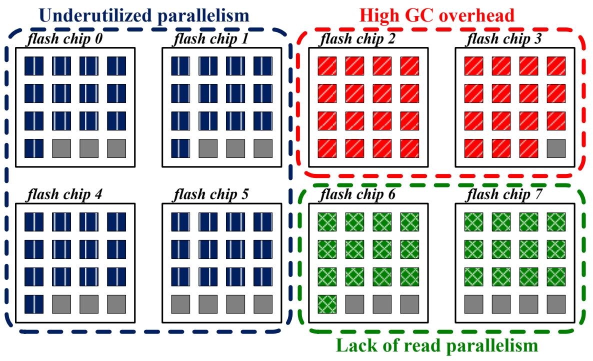


Figure 1: Normalized average read response times of seven workloads with different degrees of parallelism.

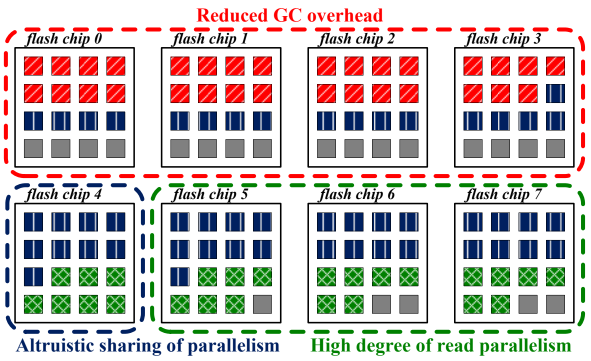
Figure 1 illustrates the effect of parallelism on the average read performance across seven workloads from Microsoft production servers [14]. Each runs alone, but by reducing the number of flash memory channels and chips available in the SSD from 4(channels)x4(chips per channel) to 1x2, workloads such as `DAP-PS`, `LM-TBE`, and `RAD-BE` experience over a two-fold increase in the average read response time.

In this paper, we make an argument for allocating resources dynamically by considering the utility of each tenant to exploit the abundant parallelism in the SSD and to mitigate the effects of I/O interferences. Our intuition is that SSDs can leverage its unique characteristics to make data placements that satisfy both performance isolation and improved efficiency without the overheads associated with load balancing and data relocation. For SSDs, the load can be balanced by controlling where the data will be written, and data relocation is natural with the use of internal management tasks such as garbage collection, wear leveling, and read scrubbing.

To that end, we propose a utilitarian performance isolation (UPI) scheme where tenant writes are striped across disjoint sets of flash memory chips to reduce I/O interference, reads are serviced from wherever the data is located, and data is relocated among one set to another when necessary. Our preliminary results show that UPI reduces the 99% QoS by 38.5% for a latency-critical tenant, and the average response time by 16.1% for a throughput-oriented one compared to the static approaches.



(a) Static partitioning of resources.



(b) Dynamic allocation in UPI.

Figure 2: Example of three tenants (blue ■, red 🟥, and green 🟩) sharing an SSD. Figure 2a illustrates static resource allocation according to capacity. Figure 2b shows how UPI allocates based on workload demand.

2 Related Work

- ***Mitigating the effects of garbage collection in SSDs.*** Resource contention between host request handling and background management tasks (such as garbage collection) has been pointed out as the main source of performance degradation in SSDs. ttFlash [21] uses a RAID-like parity scheme to reduce garbage collection (GC)-induced slowdowns; QoSFC [15] dynamically manages SSD tasks to maintain a stable performance state; and RL-assisted GC [13] schedules GC by predicting host's idle time with reinforcement learning. These techniques reduce the effects of background tasks in SSDs and complement our proposed scheme for reducing inter-tenant interferences.

- ***Static resource partitioning in SSDs.*** By dedicating separate flash memory channels or chips to each tenant, statically partitioned SSDs aim to provide deterministic I/O. This approach was first proposed in vFlash [20] and more recently has culminated to the proposal of *NVMe sets* [3]. FlashBlox [10] manages asymmetric wears across isolated regions by swapping the entire data between channels in coarse time granularities. However, partitioning resources for isolation is a double-edged sword as it also slashes the inherent parallelism in the device, and each isolated region may become underutilized or overcrowded.

- ***Data lifetime tagging in SSDs.*** Grouping data of similar lifetime reduces the number of valid page copies during GC, and thus improves its overall efficiency [5, 9]. Multi-stream [12] first proposed an interface for the host to tag data with similar lifetimes, and this has been included as *directives* in the NVM Express specification [2]. The effectiveness of multi-streaming has been demonstrated across multiple domains such as key-value store [12], virtualized storage [16], high-performance computing [8], and file system [18]. However, its performance improvements stem from the increased overall efficiency of GC, and it does not directly address performance isolation in shared SSD settings.

- ***Data placement in storage arrays.*** Several studies investigate optimal data placement in large storage arrays so that each tenant receives its fair share of I/O resources [7,17,19,22]. While these techniques consider data relocation and load balancing as performance overheads, SSDs present unique opportunities—out-of-place update and existing internal need for data relocation—that make it intuitive to handle data placement and relocation within the device.

3 Utilitarian Performance Isolation

The proposed utilitarian performance isolation (UPI) scheme decouples resources to perform write (allocated chips) and resources to store data (allocated blocks). We follow the multi-stream model [12, 16] and allow data from only one tenant to be written to each flash memory block. Thus, at any given point in time, each block is associated with only one tenant, or is unused (holds no valid data).

Each tenant is allocated a set of flash memory chips for writing data. We call this the *allocation set* S_t of a tenant t. Allocation sets are mutually exclusive and collectively exhaustive. A tenant writes data to chips of its allocation set, but may not write to a chip in another tenant's set, even if there is an unused block. Allocation sets dynamically change over time, reacting to the workload demands and state of the SSD. Reads, on the other hand, are serviced from wherever the data is located, regardless of sets. SSD internal management tasks such as garbage collection (GC) may read a tenant's data located anywhere and write them across its own set.

Figure 2 illustrates the static partitioning and our proposed UPI scheme. In this scenario, the SSD is shared among three tenants (blue 🟦, red 🟥, and green 🟩). The blue tenant requires large capacity but infrequently accesses the device, the red is a write-intensive application, and the green's read requests are QoS-sensitive. In Figure 2a, the flash memory chips are divided among the three tenants according to capacity. This results in underutilized chips for the region assigned to the blue tenant, high GC overhead for the red, and limited parallelism for the green. On the other hand, in Figure 2b, each tenant's allocation is based on its dynamic workload characteristics. Blue altruistically shares its parallelism with other tenants, the red benefits from the increased write throughput and reduced GC overhead, and the green enjoys the increased degree of parallelism and isolated performance for its read accesses.

Allocation sets thus must be carefully assigned for performance isolation while maximizing the overall parallelism. Intuitively, a throughput-oriented workload should have a larger allocation set to meet its demands, but this may harm the performance of other tenants reading from those chips. On the other hand, a QoS-sensitive tenant prefers to have its data isolated from others, but this limits the overall utilization, both in terms of capacity and parallelism. We first explain the utility function that establishes a figure of merit, and then describe the set allocation and data relocation policies.

3.1 Utility Function

A utility of a tenant is high when its reads experience less traffic. $Util(t,S)$ is the utility of tenant t given the allocation set S, and is defined as follows:

$$Util(t,S) = \frac{\sum_{c}^{chips} N_r(t,c)}{\sum_{c}^{chips} \left(\frac{N_r(t,c)}{1 - Traffic(c,S)}\right)} \tag{1}$$

Where $N_r(t,c)$ is the number of expected reads from tenant t to chip c, and $Traffic(c,S)$ is the expected traffic intensity of chip c given the allocation set S. For N_r, we use the number of observed reads in the past to estimate future reads. The denominator of the utility function is a weighted sum of reads, where the weight models the delay in an M/M/1 queue. Thus, the utility of a tenant approaches 1 (maximum) as the traffic from which it reads approaches 0 (idle).

$Traffic(c,S)$ ranges from 0 to 1 and indicates the overall busyness of the chip, and is computed as below:

$$Traffic(c,S) = \frac{\sum_{t}^{tenants} (N_r(t,c) \cdot \tau_r + N_p(t,c,S) \cdot \tau_p)}{Time_{window}} \tag{2}$$

Where τ_r is the flash memory read latency and τ_p is the program latency. Erase latency is ignored as it becomes negligible with a large number of pages per block. While we expect the number of reads in the future N_r to be similar to that of the number of observed reads, the number of expected programs N_p also depends on the allocation set S and the write amplification factor WAF, and is estimated as follows:

$$N_p(t,c,S) = \begin{cases} \frac{N_w(t) \cdot WAF(t)}{|S_t|} & \text{if } c \in S_t \\ 0 & \text{otherwise} \end{cases} \tag{3}$$

Where N_w is the number of programs due to tenant's host writes (excluding GC programs). Because GC activities can be sporadic causing wide variations in observation, we compute the number of expected programs based on N_w and WAF (approximated by the number of valid page copies during GC). We then normalize N_p by the size of the tenant's allocation set $|S_t|$ as writes are striped across chips in its set.

3.2 Set Allocation

Our objective is to find an allocation set S that minimizes the max-min ratio of utility across all tenants. This problem is difficult as it can be reduced to an NP-complete partition problem (with variables instead of fixed integers).

Instead, we approximate the solution by transferring one chip from the allocation set of the tenant with the maximum utility to that of the minimum. The rationale is because increasing the set size improves the utility of the tenant in the following ways. First, writes of the tenant are spread across a larger number of chips, reducing the traffic intensities. Second, reads are no longer affected by writes of other tenants. To prevent thrashing of chips between sets, the transfer is only made when the expected utility of a tenant losing a chip is at least as high as the current utility of the tenant gaining it. Furthermore, the chip with the least number of observed reads is transferred to reduce the inter-tenant I/O interferences.

Through set allocation, a chip that once belonged to a set of one tenant may belong to that of another. However, this does not mean that all the data must be relocated immediately. In UPI, resources to perform writes and resources to store data are decoupled, and data of one tenant may remain on the chip of another tenant, or relocate to its allocation set over time.

3.3 Data Relocation

SSDs internally handle data relocation to reclaim space (garbage collection), to provide data integrity (read

Table 1: SSD configuration.

Parameter	Value	Parameter	Value
# of channels	3	Read latency	50μs
# of chips/channel	4	Program latency	500μs
# of planes/chip	2	Erase latency	5ms
# of blocks/plane	1024	Data transfer rate	400MB/s
# of pages/block	512	Physical capacity	192GB
Page size	16KB	Logical capacity	150GB

Table 2: Tenant's workload characteristics.

	DAS-AS	DTRS	LM-TBE
Num. of writes (m)	0.1	5.8	9.2
Num. of reads (m)	1.3	12.0	34.7
Avg. wr. size (KB)	7.2	31.9	61.9
Avg. rd. size (KB)	31.5	21.8	53.2
Peak write IOPS	53.2	408.8	589.1
Peak read IOPS	89.9	745.3	4173.8

scrubbing), and to prolong storage lifetime (wear leveling). We enhance these existing mechanisms to handle data relocation for performance isolation.

This can be achieved by considering the number of reads for a block if the chip belongs to another tenant. By taking into account the number of reads in the victim selection's cost-benefit analysis, frequently read data is relocated back to its set, isolating its data from other I/O activities. Infrequently accessed cold data may remain in another set and need not be actively moved.

4 Evaluation

4.1 Experimental Setup

We implement the proposed UPI scheme on top of the DiskSim environment [1] by enhancing its SSD extension [4]. We construct a modest SSD with 3 channels and 4 chips per channel, and a 150GB logical and 192GB physical capacity (28% over-provisioned). Table 1 summarizes the SSD configuration used in our experiments. Host request handling operates in a non-blocking manner to fully utilize the underlying parallelism, and garbage collection (GC) acts as the mechanism for data relocation among sets. Host writes and GC data are written to different blocks as a means to separate hot and cold data. We use a simple priority scheduler at the flash memory subsystem: host requests have precedence over GC requests.

We use three workloads, `DAS-AS`, `DTRS`, and `LM-TBE` from Microsoft production server traces [14] to model three tenants with distinct access patterns. `DAS-AS` has the lowest throughput, but has the highest read-to-write ratio—this is the tenant that wants to be isolated. On the other hand, `DTRS` is a relatively random workload with bursts of writes and high GC overheads, and `LM-TBE` sequentially accesses the device with high intensity—these represent tenants that cause I/O interferences. The workload characteristics are summarized in Table 2.

All workloads run to completion (about 24 hours), and the address of each request is modified to fit into a 50GB range for each tenant. Prior to replaying the traces, data is randomly written to physical locations in each tenant's allocation set to emulate a pre-conditioned state.

4.2 Preliminary Results

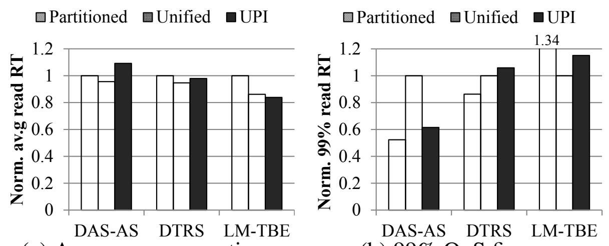
UPI is evaluated against two static allocation schemes:

Partitioned dedicates separate chips to each tenant and completely isolates performance.

Unified shares all resources among tenants to maximize utilization and parallelism.

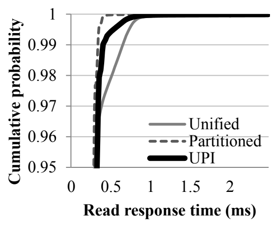
Figure 3 compares the performance of UPI against the Partitioned and Unified scheme. We first investigate the average performance shown in Figure 3a. Compared to the Partitioned scheme, UPI reduces the average read response time for `LM-TBE`, a throughput-oriented tenant, by 16.1%. The Unified scheme achieves similar an improvement with a 13.8% reduction. The average performance of UPI for `DAS-AS` is slightly worse than the Partitioned scheme, as the low write intensity of `DAS-AS` compacts its data to a smaller set. For the 99% QoS performance of reads in Figure 3b, UPI improves by 38.5% for `DAS-AS`, a latency-sensitive tenant, compared to the Unified scheme. For the same workload, we also observe that the Partitioned scheme improves by 47.7% for the 99% QoS figure.

Although the Partitioned scheme achieves perfect isolation for `DAS-AS`, this comes at a cost of performance degradation for `LM-TBE` in both the average response time and QoS figure. On the other hand, the Unified scheme performs well for `LM-TBE`, as it dominates the overall traffic across all chips, but this severely degrades the QoS performance for `DAS-AS`. Our proposed UPI scheme finds a balance between the two extremes: it achieves better average performance for high-throughput tenants compared to the Partitioned scheme, and better QoS performance for latency-critical tenants compared

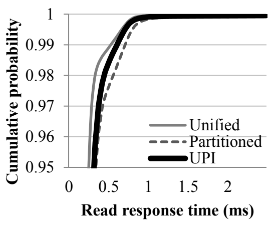


(a) Average response time. (b) 99% QoS figure.

Figure 3: Performance of the Partitioned, the Unified, and UPI.



(a) CDF for `DAS-AS`.

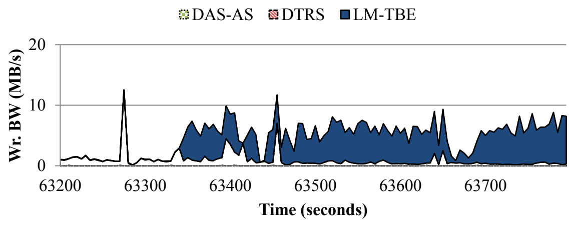


(b) CDF for `LM-TBE`.

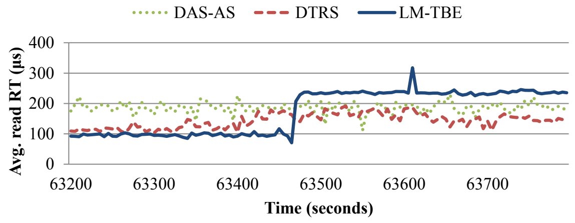
Figure 4: Response time CDFs for Unified, Partitioned, and UPI under workloads `DAS-AS` and `LM-TBE`.

to the Unified. This property of UPI can also be observed in the response time CDFs in Figure 4.

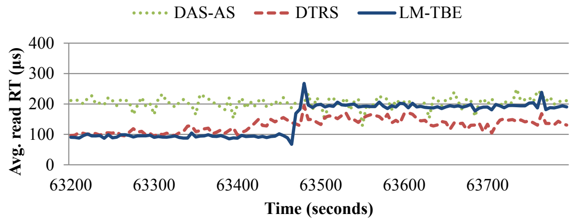
We microscopically examine the performance of the Partitioned scheme and UPI in Figure 5. Figure 5a shows the average read response time sampled every 5 seconds during a 600-second window, approximately 17 hours into the workload (when `LM-TBE` starts to increase its write bandwidth). The performance of the Partitioned scheme in Figure 5b is affected by this, resulting in the overall increase in response time. On the other hand, UPI in Figure 5c dynamically increases the allocation set for `LM-TBE` to balance the traffic across chips, thus achieving better performance.



(a) Write bandwidth.



(b) Average response time of the Partitioned scheme.



(c) Average response time of UPI.

Figure 5: Increase in `LM-TBE` write bandwidth (Figure 5a) causes performance degradation for the Partitioned scheme (Figure 5b). In comparison, UPI (Figure 5c) is less affected by this.

4.3 Discussions

In this work, UPI considers all tenants equally, but the utility function in Equation 1 can be fine-tuned to provide differentiated services among tenants. By taking an exponent to $1 - Traffic(c, S)$, we can modify the degree in which traffic intensities affect its utility, thereby providing asymmetric performances to tenants.

Similarly, UPI can tradeoff between the average and tail latency performance. By applying another exponent value to $N_r(t, c)$ of both the numerator and the denominator, the utility function can shift its focus between the common case and the rare.

By relaxing the mutually exclusive and collectively exhaustive property of allocation sets, UPI can cover other allocation policies. For example, one such policy can exclude a particular chip from any set to only allow reads to be serviced from it.

Furthermore, the data relocation policy can be adjusted between eager (stronger isolation) and lazy (better efficiency). By scaling down the degree in which the number of reads for a block affects the victim selection, only heavily-read data in another tenant's set will be moved. Scaling it up will cause data to be relocated aggressively, albeit at a cost of higher GC overhead.

5 Conclusion

In this paper, we presented a utilitarian performance isolation (UPI) scheme for multiple tenants sharing a single SSD. UPI considers the utility of each tenant and dynamically allocates resources so that the inherent parallelism of SSDs can be fully exploited all the while mitigating the effects of inter-tenant I/O interferences. Our preliminary results are promising, with UPI reducing the 99% QoS performance by 38.5% for a latency-sensitive workload, and the average performance by 16.1% for a high-throughput workload. Our design can be extended in several directions, such as providing asymmetric services to tenants and exploring the tradeoff between the average and tail latency performance.

Acknowledgment

We thank the anonymous reviewers for their constructive and insightful comments, and also thank Sang Lyul Min, Jongmoo Choi, and Eunji Lee for reviewing the early stages of this work. This work was supported in part by SK Hynix and the National Research Foundation of Korea under the PF Class Heterogeneous High Performance Computer Development (NRF-2016M3C4A7952587). Institute of Computer Technology at Seoul National University provided the research facilities for this study.

References

[1] The disksim simulation environment version 4.0 reference manual (cmu-pdl-08-101). `http://www.pdl.cmu.edu/PDL-FTP/DriveChar/CMU-PDL-08-101.pdf`, 2008. Parallel Data Laboratory.

[2] NVM Express revision 1.3. `http://nvmexpress.org/wp-content/uploads/NVM_Express_Revision_1.3.pdf`, 2017. NVM Express.

[3] Solving latency challenges with NVM express SSDs at scale. `https://www.flashmemorysummit.com/English/Collaterals/Proceedings/2017/20170809_SIT6_Petersen.pdf`, 2017. Flash Memory Summit.

[4] AGRAWAL, N., PRABHAKARAN, V., WOBBER, T., DAVIS, J. D., MANASSE, M. S., AND PANIGRAHY, R. Design tradeoffs for SSD performance. In *USENIX Annual Technical Conference* (2008).

[5] CHIANG, M.-L., LEE, P. C., CHANG, R.-C., ET AL. Using data clustering to improve cleaning performance for flash memory. *Software-Practice & Experience 29*, 3 (1999), 267–290.

[6] DEAN, J., AND BARROSO, L. A. The tail at scale. *Communications of the ACM 56*, 2 (2013), 74–80.

[7] GULATI, A., SHANMUGANATHAN, G., AHMAD, I., WALDSPURGER, C., AND UYSAL, M. Pesto: online storage performance management in virtualized datacenters. In *ACM Symposium on Cloud Computing (SoCC)* (2011).

[8] HAN, J., KOO, D., LOCKWOOD, G. K., LEE, J., EOM, H., AND HWANG, S. Accelerating a burst buffer via user-level I/O isolation. In *IEEE International Conference on Cluster Computing (CLUSTER)* (2017).

[9] HSIEH, J.-W., KUO, T.-W., AND CHANG, L.-P. Efficient identification of hot data for flash memory storage systems. *ACM Transactions on Storage (TOS) 2*, 1 (2006), 22–40.

[10] HUANG, J., BADAM, A., CAULFIELD, L., NATH, S., SENGUPTA, S., SHARMA, B., AND QURESHI, M. K. Flashblox: Achieving both performance isolation and uniform lifetime for virtualized SSDs. In *USENIX Conference on File and Storage Technologies (FAST)* (2017).

[11] JUN, B., AND SHIN, D. Workload-aware budget compensation scheduling for NVMe solid state drives. In *IEEE Non-Volatile Memory System and Applications Symposium (NVMSA)* (2015).

[12] KANG, J.-U., HYUN, J., MAENG, H., AND CHO, S. The multi-streamed solid-state drive. In *USENIX Workshop on Hot Topics in Storage and File Systems (HotStorage)* (2014).

[13] KANG, W., SHIN, D., AND YOO, S. Reinforcement learning-assisted garbage collection to mitigate long-tail latency in SSD. *ACM Transactions on Embedded Computing Systems (TECS) 16*, 5s (2017), 134.

[14] KAVALANEKAR, S., WORTHINGTON, B., ZHANG, Q., AND SHARDA, V. Characterization of storage workload traces from production Windows servers. In *IEEE International Symposium on Workload Characterization (IISWC)* (2008).

[15] KIM, B. S., AND MIN, S. L. QoS-aware flash memory controller. In *IEEE Real-Time and Embedded Technology and Applications Symposium (RTAS)* (2017).

[16] KIM, J., LEE, D., AND NOH, S. H. Towards SLO complying SSDs through OPS isolation. In *USENIX Conference on File and Storage Technologies (FAST)* (2015).

[17] PARK, N., AHMAD, I., AND LILJA, D. J. Romano: Autonomous storage management using performance prediction in multi-tenant datacenters. In *ACM Symposium on Cloud Computing (SoCC)* (2012).

[18] RHO, E., JOSHI, K., SHIN, S.-U., SHETTY, N. J., HWANG, J., CHO, S., LEE, D. D., AND JEONG, J. Fstream: Managing flash streams in the file system. In *USENIX Conference on File and Storage Technologies (FAST)* (2018).

[19] SHUE, D., FREEDMAN, M. J., AND SHAIKH, A. Performance isolation and fairness for multi-tenant cloud storage. In *USENIX Symposium on Operating Systems Design and Implementation (OSDI)* (2012).

[20] SONG, X., YANG, J., AND CHEN, H. Architecting flash-based solid-state drive for high-performance I/O virtualization. *IEEE Computer Architecture Letters 13*, 2 (2014), 61–64.

[21] YAN, S., LI, H., HAO, M., TONG, M. H., SUNDARARAMAN, S., CHIEN, A. A., AND GUNAWI, H. S. Tiny-tail flash: Near-perfect elimination of garbage collection tail latencies in NAND SSDs. In *USENIX Conference on File and Storage Technologies (FAST)* (2017).

[22] YANG, Z., HOSEINZADEH, M., ANDREWS, A., MAYERS, C., EVANS, D. T., BOLT, R. T., BHIMANI, J., MI, N., AND SWANSON, S. Autotiering: Automatic data placement manager in multi-tier all-flash datacenter. In *IEEE International Performance Computing and Communications Conference (IPCCC)* (2017).