WA-OPShare: Workload-Adaptive Over-Provisioning Space Allocation for Multi-Tenant SSDs

Yuhong Wen, You Zhou, Fei Wu, *Member, IEEE*, Shu Li, Zhenghong Wang, and Changsheng Xie

***Abstract*—Sharing a flash-based solid-state drive (SSD) among multiple tenants has become a common practice to improve storage utilization and cost efficiency. Meanwhile, how to allocate limited storage resources, especially the over-provisioning space (OPS) resources, among competitive tenants has emerged as a critical problem. The OPS refers to additional user-invisible storage space, whose size influences garbage collection (GC) efficiency. Due to unawareness of workload characteristics of different tenants, prior studies on multitenant OPS allocation lead to suboptimal SSD performance. In this article, we propose a novel workload-adaptive OPS allocation scheme for multitenant SSDs, called *WA-OPShare*. It targets an OPS sharing scheme that dynamically allocates the OPS among tenants to improve overall SSD performance. Two models are developed to identify underutilized storage space and predict the OPS-induced performance benefit of each tenant, respectively. Guided by the models, WA-OPShare regularly releases the underutilized storage space and then reallocates it to the tenant who can benefit the most. Experimental results show that compared to the traditional Partition and Sharing schemes, WA-OPShare improves the performance by up to 40.3% and 31.2%, and reduces the write amplification by up to 37.0% and 17.5%, respectively.**

***Index Terms*—Multitenant, over-provisioning space (OPS), solid-state drives (SSDs), virtualization, workload characteristic.**

I. Introduction

With the fast development of NAND flash technologies (e.g., 3-D stacking and multibit per cell), flash-based solid-state drives (SSDs), built with large capacities (e.g., several terabytes) and high bandwidth (e.g., a few GB/s), are widely deployed in cloud, enterprise, and mobile storage systems [1]. To improve storage utilization and cost efficiency, sharing a single SSD among multiple tenants has become a common practice [2], [3]. For example, a 4-TB SSD can hold several database instances, whose capacity is tens of to a few hundreds of gigabytes [4].

When multiple tenants run concurrently on an SSD, they compete for limited storage resources, including bandwidth, flash parallel units, capacity, and so on. Arbitrary and unconstrained competition for these limited resources will cause performance interference among tenants and low resource utilization [5], [6]. Intuitively, each tenant contends for SSD service time (e.g., of accessing flash parallel units) to process its I/O requests. It has been well studied how to allocate service time among tenants efficiently, which can borrow wisdom from scheduling approaches in traditional storage systems [6], [7], [8], [9].

However, a multitenant SSD suffers from another unique resource competition problem on the over-provisioning space (OPS). Due to the erase-before-write feature of flash memory, SSDs conduct out-of-place updates, where data are written to new flash pages while stale flash pages are invalidated. Garbage collection (GC) is required to reclaim invalid pages by migrating valid pages in victim flash blocks and then erases the blocks, causing the *write amplification* problem. GC operations may block host I/Os and seriously degrade the performance and lifetime of SSDs. The OPS refers to additional user-invisible flash storage space (e.g., 25% of SSD capacity). Increasing the OPS size can improve the GC efficiency (i.e., accommodating more invalid pages and reducing the migrations of valid pages) and, thus, indirectly enhance SSD performance and lifetime [10]. Tenants vary in their benefits of OPS resources, for example, given the same amount of OPS, tenants with different write patterns differentiate in performance gains. Hence, similar to SSD service time, the OPS is also a critical storage resource in multitenant SSDs. Unregulated allocation of OPS resources may lead to suboptimal SSD performance.

Prior studies either overlook the OPS competition problem and make tenants freely share the whole OPS of SSDs [6], [11], [12], which we refer to as the *Sharing* scheme, or employ a naive scheme, called *Partition*, that statically partitions the OPS among tenants [1], [13], [14]. The Sharing scheme incurs unpredictable performance interference between tenants, while the Partition scheme provides spatial isolation. They have a common drawback of suboptimal SSD performance. Taking different and variable workload characteristics running concurrently by tenants into consideration,

Manuscript received 3 August 2022; accepted 3 August 2022. Date of current version 24 October 2022. This work was supported in part by the National Natural Science Foundation of China under Grant 61902137, Grant U2001203, Grant 61821003, and Grant 61872413; in part by the Alibaba Group through Alibaba Innovative Research Program; and in part by the Fundamental Research Funds for the Central Universities, HUST under Grant 2021XXJS108 and Grant YCJJ202202002. This article was presented at the International Conference on Hardware/Software Codesign and System Synthesis (CODES + ISSS) 2022 and appeared as part of the ESWEEK-TCAD special issue. This article was recommended by Associate Editor A. K. Coskun. *(Corresponding author: You Zhou.)*

Yuhong Wen is with the Wuhan National Laboratory for Optoelectronics and the School of Computer Science and Technology, Huazhong University of Science and Technology, Wuhan 430074, China (e-mail: wenyuhong@hust.edu.cn).

You Zhou is with the School of Computer Science and Technology, Huazhong University of Science and Technology, Wuhan 430074, China (e-mail: zhouyou2@hust.edu.cn).

Fei Wu and Changsheng Xie are with the Wuhan National Laboratory for Optoelectronics, Huazhong University of Science and Technology, Wuhan 430074, China (e-mail: wufei@hust.edu.cn; cs_xie@hust.edu.cn).

Shu Li and Zhenghong Wang are with the Alibaba Cloud Intelligence-Infrastructure Business Unit, Alibaba Group, Hangzhou 310052, China (e-mail: s.li@alibaba-inc.com; zhengyong.wzy@alibaba-inc.com).

Digital Object Identifier 10.1109/TCAD.2022.3199966

both the Partition and Sharing schemes cannot be adaptive to the workloads and, thus, fail to maximize the benefits of valuable OPS resources, as revealed in Section II-B.

In this article, we present *WA-OPShare*, a workload-adaptive OPS allocation scheme for an SSD shared by multiple tenants. WA-OPShare is designed to dynamically allocate OPS resources among tenants to maximize overall SSD performance. To achieve this goal, we propose two key models: 1) a *space efficiency model* to identify the underutilized storage space (i.e., invalid flash pages that have not been erased for a long time) and 2) an *OPS marginal gain model* to assess the OPS-induced performance benefit of each tenant based on its workload characteristics. Periodically, WA-OPShare compacts the underutilized space to release free OPS and then workload adaptively reallocates it to the tenant who can benefit the most. These two models complement each other to realize the dynamic allocation of OPS resources among tenants and ultimately improve the overall performance of the SSD. We verify WA-OPShare using extensive trace-driven simulations. The experimental results show that compared to the traditional Partition and Sharing schemes, WA-OPShare can improve the performance by up to 40.3% and 31.2%, and reduce the write amplification by up to 37.0% and 17.5%, respectively.

II. Background and Motivation

A. Background

1) Multitenant SSDs: Modern SSDs are built with a highly parallel architecture for large capacity and high bandwidth. An SSD contains multiple channels connecting several flash dies, which are the smallest parallel unit for independent access. Each die contains thousands of flash blocks (the flash erase unit) and each block is composed of many flash pages (the flash read/write unit). Multitenancy is facilitated by not only the parallel architecture but also interface techniques. The PCIe single root I/O virtualization (SR-IOV) interface can virtualize a physical device into multiple virtual devices [15]. The NVMe interface protocol can format an SSD into several independent namespaces and offer multiple queues to dispatch I/O commands [16].

SSDs usually employ *superblock*-based flash management, where each superblock refers to a group of flash blocks with the same offset across many channels and chips [17], [18]. A superblock is the basic unit of space allocation for data writes and GC operations. SSDs generally adopt the *greedy GC* strategy which erases a superblock with the most invalid pages to obtain a free superblock. Before erasing a superblock, the valid pages therein must be migrated to other superblocks, causing *write amplification* problem. The ratio of total flash writes (including these extra flash writes generated by GC operations and so on) to the host writes is the write amplification factor (WAF), which is an important factor affecting the performance and lifetime of SSDs. An SSD can reduce its WAF by providing more OPS resources.

2) I/O Scheduling in Multitenant SSDs: In a multitenant SSD, I/O requests from tenants compete for SSD service time (e.g., to access flash dies). How to allocate the time resource and schedule I/O requests between tenants has been well studied. For example, to provide performance isolation and fairness, three kinds of schemes have been proposed: 1) budget-based I/O processing [11], [13], [19]; 2) fair queuing-based I/O scheduling [7], [8], [9], [20]; and 3) heuristic I/O scheduling [6].

In this article, we simply adopt the budget-based I/O scheduler [13] to allocate SSD service time among tenants (based on their weights) for processing their I/O requests. At the start of each epoch, each tenant is allocated a fixed I/O budget according to its weight. Whenever an I/O request is served, a certain amount of the I/O budget is consumed according to its type (read or write) and size. Once a tenant consumes all its budget in the current epoch, its subsequent I/O requests will be throttled until the budget is replenished in the next epoch. Other I/O scheduling algorithms can also be adopted, which are orthogonal to our work (focusing on OPS allocation among multiple tenants).

3) OPS Allocation in Multitenant SSDs: Besides service time competition, multitenant SSDs also suffer from storage space competition. The allocation of space resources among multiple tenants, especially OPS resources, affects the WAF of each tenant and the overall performance of the SSD. Existing OPS allocation schemes can be divided into two categories: 1) hard allocation and 2) soft allocation.

Hard Allocation: Hard allocation schemes allocate separate parallel storage components (e.g., flash channels [21] or dies [1], [13]) to each tenant. Physical isolation of allocated OPS prevents the performance interference problem and provides strong isolation among multiple tenants. However, they have several disadvantages, such as low maximum parallel bandwidth for each tenant, being nonconductive to die-level RAID protection [22], inflexible and large OPS allocation granularity [1], and nonpreemptive OPS allocation (which prevents the optimization of OPS efficiency).

Soft Allocation: Soft allocation schemes allocate a group of flash blocks/superblocks spanning many dies to each tenant and allow a flash die to be shared by all tenants. Data of different tenants are written into different flash superblocks, ensuring that each superblock stores data of no more than one tenant. Therefore, the tenant who triggers the GC process only migrates its own data, avoiding the performance interference problem caused by the GC process. In this article, we adopt the soft allocation scheme which enables a flexible OPS allocation and provides a higher parallel bandwidth for each tenant than the hard allocation scheme.

Most soft allocation schemes fall into two categories: 1) the *Partition* [1], [13], [14] and 2) the *Sharing* [6], [11], [12] schemes. The Partition scheme divides superblocks of the SSD into different superblock groups, each is allocated to a tenant and adopts the *local GC* strategy. Each tenant only erases its own superblocks and returns erased superblocks to its own superblock group. Thus, the Partition scheme ensures fixed OPS resources for tenants and prevents a tenant from unfairly occupying excessive OPS resources. The Sharing scheme allows tenants to freely compete for OPS resources in the shared superblock pool, and uses the *global GC* strategy. It chooses superblocks regardless of their affiliation to tenants with a unified policy (e.g., the greedy policy) to erase and maintains a shared free superblock pool. Unfortunately, both the Partition and Sharing schemes are all unable to realize an efficient and workload-adaptive OPS allocation and finally result in suboptimal SSD performance (see Section II-B). In addition, OPS-Isolation [23] proposes to partition the OPS among tenants dynamically to achieve proportional bandwidth fairness. Different from this work, we aim to maximize the benefit of OPS resources in this article, which requires identifying underutilized OPS resources and modeling the gain of allocating OPS to each tenant.

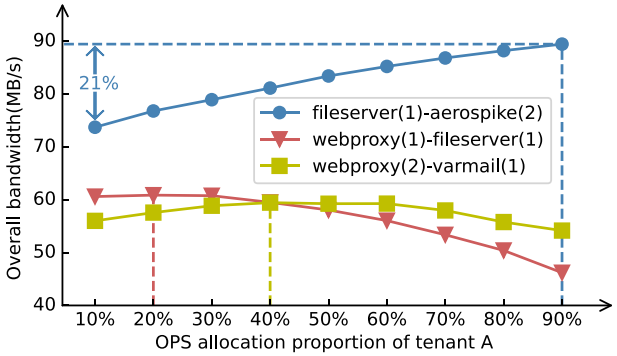

Fig. 1. Overall bandwidth of the Partition scheme under different OPS allocation configurations between two tenants working concurrently in the simulated SSD. "$X(a) - Y(b)$" means tenant A runs the workload X with weight a and tenant B runs the workload Y with weight b.

B. Motivation

We conduct experiments on the SSDsim simulation platform [24] to study: 1) the drawbacks of the Partition and Sharing schemes and 2) the performance impacts of OPS resources under different workloads. More details about simulator parameters and workload characteristics are shown in Section IV. If not specified, the simulated SSD contains 32-GB logical capacity and 8-GB OPS resources by default.

1) Partition Scheme Cannot Adapt to Workloads: The Partition scheme prevents unfairly competition for OPS resources among tenants, however, losing the opportunity to achieve a higher performance by workload adaptively adjusting OPS allocation. We conduct experiments to demonstrate the performance benefit this opportunity can bring. As shown in Fig. 1, an allocation of 90% OPS resources to workload "fileserver" and 10% to workload "aerospike" achieves the optimal overall performance in the Partition scheme. There is a 21% performance gap between the worst performing OPS allocation configuration (workload fileserver occupies 10% of OPS resources) and the optimal-performing OPS allocation configuration (workload fileserver occupies 90% of OPS resources), which motivates us to dynamically remain the optimal allocation for the optimal performance.

Unfortunately, this optimal allocation is not constant and not applicable to different workloads. As shown in Fig. 1, the optimal OPS allocation ratio is 90%, 20%, and 40%, respectively, in workload groups of "fileserver(1)-aerospike(2)," "webproxy(1)-fileseerver(1)," and "webproxy(2)-varmail(1)," since the performance improvement by OPS resources differentiates in different workloads. Considering numerous influencing factors (workload characteristics, read/write ratio, weight, and so on) and their variability, it is impractical to manually assign the optimal OPS allocation. In summary, the static Partition scheme cannot adapt to the dynamic change of workloads.

2) Sharing Scheme Lacks Wise OPS Allocation Guiding: Although the Sharing scheme allows tenants to freely compete for all OPS resources, it fails to maximize the benefit of OPS resources, leading to poor performance, as shown in Fig. 2. Each of the two running workloads is allocated a portion of initial OPS resources based on its weight ratio, i.e., the workloads "webproxy" and "varmail" are, respectively, allocated (2/3) and (1/3) of OPS resources. During the former 150 min, the performance isolation is guaranteed by the Partition scheme; thus, the WAF change of varmail between 100th min to 150th min does not affect the WAF of webproxy, as shown in Fig. 2(a). However, there is no performance isolation between two workloads during the later 150 min because the Sharing scheme does not limit the competition among workloads, which introduces performance interference. More importantly, the overall average bandwidth in the Sharing scheme is 20.6% worse than it in the Partition scheme due to the lack of wise OPS allocation guidance (resulting in more migration overhead during GC), as shown in Fig. 2(b) and (d).

Fig. 2(c) demonstrates that the uncontrolled OPS allocation of the Sharing scheme results in poor performance. In the former 150 min, since the Partition scheme guarantees performance isolation, the occupied OPS size of each workload remains unchanged. Then, from the 150th min, the workload with a larger WAF (varmail) starts to preempt the OPS resources of the workload with a smaller WAF (webproxy) due to the Sharing scheme. This competition result originates from the global GC strategy it adopts. It chooses the flash superblock with the most invalid pages to reclaim, which minimizes the GC migrating overhead and the WAF. Usually, a tenant with more invalid pages in its superblocks shows a lower WAF. Therefore, superblocks of the tenant with a lower WAF are more likely to be reclaimed and preempted by other tenants with higher WAFs until their WAFs are the same. However, the performance gap between the Sharing and Partition schemes shows that keeping the same WAF among tenants does not maximize overall performance, which motivates us to design a wiser dynamic OPS allocation scheme.

3) Performance Benefit of OPS Resources Varies in Different Workloads: Previous experiment results prove that the Partition and Sharing schemes cannot achieve an efficient allocation of OPS resources in the multitenant environment. To workload-adaptively allocate OPS resources for better SSD performance, we need to know the benefit of them for each tenant. We use the marginal gain (MG) to characterize OPS benefits, defined as follows: $\mathrm{MG}(x, y) = \mathrm{Bandwidth}(x + y) - \mathrm{Bandwidth}(x)$. $\mathrm{Bandwidth}(x)$ and $\mathrm{Bandwidth}(x + y)$ is the bandwidth achieved with x GB and $x + y$ GB OPS resources separately, and $\mathrm{MG}(x, y)$ is the bandwidth increase that additional y GB OPS resources can bring.

OPS resources help decrease GC migrating overhead and WAF, thus indirectly improving SSD performance. We first demonstrate the impact of write patterns on OPS benefits, as shown in Fig. 3(a). From experimental results, we can find that: 1) the $\mathrm{MG}(x, 1)$ curve varies in the three workloads with different write patterns and 2) under the same write pattern, the $\mathrm{MG}(x, 1)$ decreases as OPS resources increases, exhibiting diminishing marginal utility (the exception MG(2,1) in "webproxy_w" may be occasional boosts manifested in a special state).

Besides the write patterns and the current OPS size, the write ratio also infects OPS benefits, as shown in Fig. 3(b). The experimental results of workloads with different write ratios also show that the more OPS resources the workload currently has, the lower the benefit from more OPS resources. In addition, when current OPS resources are scarce (i.e., 1 GB), an additional 1 GB of OPS resources can deliver significant performance gains. Tenants with a larger read ratio have a larger improvement due to the asymmetry that the read speed of flash is usually a dozen times faster than the write speed.

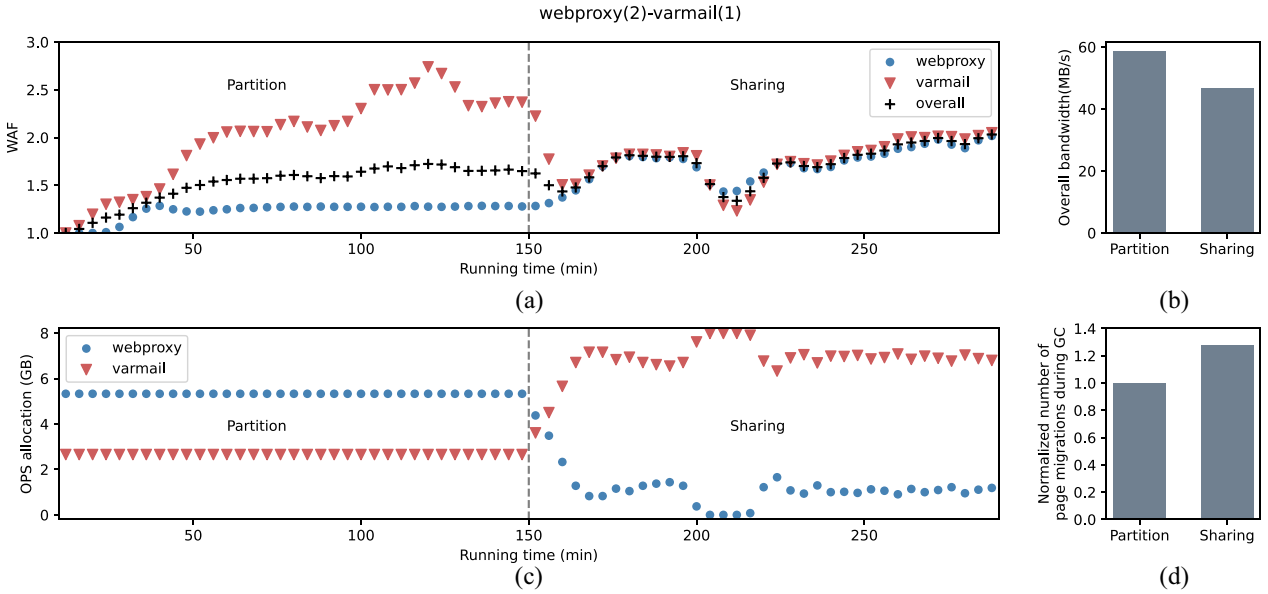

Fig. 2. (a) WAF change at runtime, (b) bandwidth comparison, (c) OPS competition at runtime, and (d) GC overhead comparison of the Partition and Sharing schemes. The experiment adopts the Partition scheme in the former 150 min and adopts the Sharing scheme in the later 150 min.

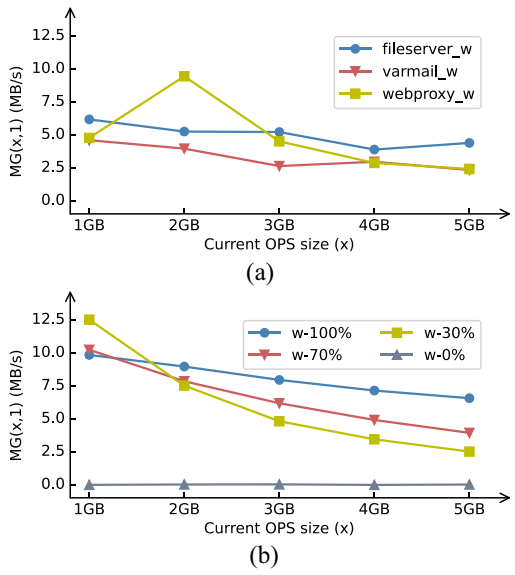

Fig. 3. Marginal gain of OPS resources in workloads with (a) different write patterns and (b) different write ratios. The *y*-axis coordinate MG(x,1) means the bandwidth increase that additional 1GB OPS can bring when the workload has allocated xGB OPS. The "fileserver_w," "varmail_w," and "webproxy_w" are the pure-write workload filtered from the workload fileserver, varmail, and webproxy, respectively. "w-x%" is the generated random 4-kB IO workload with x% write ratio.

However, with the gradual increase of OPS resources, the impact of this asymmetry gradually decreases. Write-intensive workloads have strong demand for OPS resources, thus the workload with larger write ratios has a higher marginal gain. In particular, pure-read workload ("w-0%") hardly benefits from OPS resources.

In conclusion, the performance benefit of OPS resources increase for a tenant dynamically changes depending on its write pattern, read/write ratio, and the current OPS resources it owns, which is a challenge for us to predict the performance benefit of OPS resources at runtime.

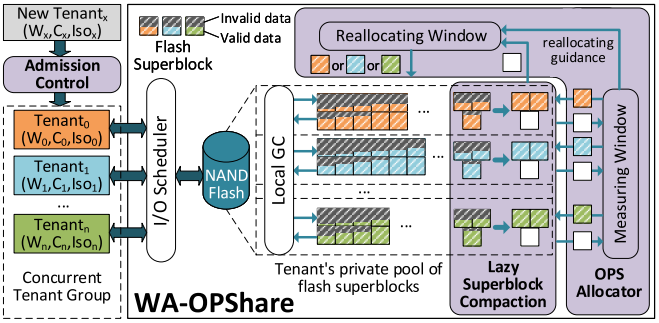

Fig. 4. Architecture of WA-OPShare.

III. Design

A. Overview of WA-OPShare

To maximize the performance gain from OPS resources, we present a novel scheme, called *WA-OPShare*, to allocate OPS resources among multiple tenants in a workload-adaptive manner, as shown in Fig. 4.

When admitted by the SSD, each tenant is associated with a predefined service level objective (SLO), including the weight *W*, capacity *C* and isolation level *Iso*. Only when *C* is no more than the remaining capacity of the shared SSD, the new *Tenant* is admitted to the concurrent tenant group. WA-OPShare simply adopts the existing budget-based I/O scheduler [13] (as described in Section II-A2) to fairly allocate time budgets to tenants based on their weights, and divides flash superblocks into different groups. Each tenant is allocated a separate group of superblocks as its private superblock pool and the proportional initial budget and OPS resources based on its weight *W*. Tenants can only write data and perform local GC in their own superblock pools, which provides isolation among tenants. When *Iso* is set as 0, it represents the *Tenant* does not require the isolation guarantee; thus, WA-OPShare dynamically adjusts its OPS resources for better overall SSD performance. Otherwise, when *Iso* is set as 1, WA-OPShare ensures that resources of the *Tenant* remain unchanged to guarantee its performance isolation.

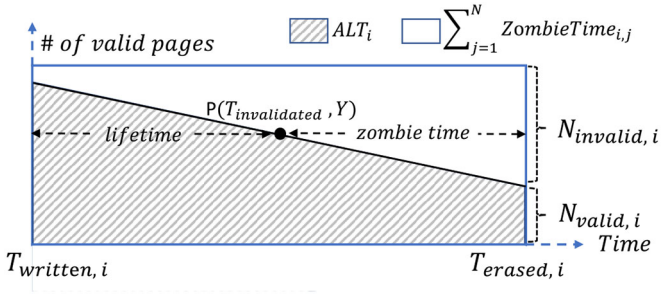

Fig. 5. Lifetime and zombie time of data pages in superblock i.

WA-OPShare consists of two key modules, the *lazy superblock compaction* and the *OPS allocator*. The lazy superblock compaction module periodically identifies lazy superblocks (defined in Section III-B) from all tenants' private superblock pools, and compacts valid pages of N lazy superblocks into $N-1$ superblocks to release a free superblock which is delivered to the OPS allocator module for reallocation. The OPS allocator module alternates between the *measuring window* and the *reallocating window*. In the measuring window, the released free superblock is returned to its original tenant to measure the marginal gain of this tenant for more OPS resources, while in the reallocating window, the released free superblock will be allocated to the tenant with the most marginal gain for better overall SSD performance. These two modules complement each other; thus, WA-OPShare can dynamically adjust OPS allocation in a workload-adaptive manner to maximize the benefit of OPS resources and improve the SSD performance.

To realize efficient compaction and reallocation of OPS resources, we develop two models: 1) a *space efficiency* model to identify lazy superblocks (in Section III-B) and 2) an *OPS marginal gain* model to determine the target tenant for OPS reallocation (in Section III-C).

B. Lazy Superblock Compaction

Considering the access locality and variability of workloads, there may be flash superblocks that have not been erased for a long time and contain many long-lived invalid pages, called *lazy superblocks*. Those long-lived invalid pages leave part of the space underutilized and result in low GC efficiency and suboptimal SSD performance in the long term. The lazy superblock compaction module monitors the space efficiency of superblocks at runtime and releases these long-lived invalid pages in time by compacting valid pages of lazy superblocks together to improve space efficiency.

Specifically, each flash page P of superblock i goes through three operation states: being written ($T_{\text{written},i}$), then invalidated ($T_{\text{invalidated}}$), and finally erased ($T_{\text{erased},i}$), as shown in Fig. 5. The point $P(T_{\text{invalidated}}, Y)$ means the page P is invalidated at time $T_{\text{invalidated}}$ and there are Y valid pages in the superblock at that time. We refer to the time period length between being written and being invalidated as the *lifetime* of page P, and the time period length between being invalidated and being erased as the *zombie time*. We regard the accumulation of zombie time ($\sum_{j=1}^{N} \text{ZombieTime}_{i,j}$) as the *Laziness* of superblock i which is composed of N flash pages. Thus, the *Laziness* measures how much time and space invalid pages in a superblock have "wasted" and is inversely proportional to the utilization efficiency of storage space. Considering the ratio of valid pages ($\text{Ratio}_{\text{valid},i}$) of the lazy superblock i, the lazy superblock compaction module periodically compact superblocks with the least *Efficiency* by the *space efficiency* model shown in

$$\text{Efficiency}_i = \frac{\text{Ratio}_{\text{valid},i}}{\text{Laziness}_i} = \frac{N_{\text{valid},i}}{N \times \sum_{j=1}^{N} \text{ZombieTime}_{i,j}} \quad (1)$$

$$\sum_{j=1}^{N} \text{ZombieTime}_{i,j} = N \times \left(T_{cur,i} - T_{\text{written},i}\right) - \text{ALT}_i. \quad (2)$$

To reduce the metadata overhead, WA-OPShare only maintain two 4-(byte) metadata parameters ($T_{\text{written},i}$ and ALT_i) for each superblock by (2) instead of maintaining the $\text{ZombieTime}_{i,j}$ for each page. $T_{\text{written},i}$ is the time when superblock i is fully written, and ALT_i is the accumulated lifetime of the superblock i. Whenever a page P of superblock i is invalidated, ALT_i increases by its lifetime (the length between $T_{\text{written},i}$ and its invalidated time $T_{\text{invalidated},i}$). For $N_{valid,i}$ valid pages, each increases ALT_i by the length between $T_{\text{written},i}$ and $T_{cur,i}$. $T_{cur,i}$ is the current time calculating Efficiency_i and does not need to be recorded additionally. Specifically, for a 40-GB SSD emulated in our experiments, where the superblock size is 16 MB (see Table II), the additional metadata size, i.e., DRAM space overhead, is only 20 kB. Assuming a 1TB SSD with 256-MB superblock size, the extra DRAM requirement would be as small as 32 kB. The compaction of lazy superblocks will generate the migration of valid pages and consume SSD bandwidth, however, it also releases free OPS to improve space efficiency and SSD bandwidth. We empirically limit the bandwidth consumed by the lazy superblock compaction module to less than 1% of SSD bandwidth, while enjoying the performance gains it brings.

C. OPS Allocator

To maximize the benefit of superblocks released from the lazy superblock compaction module, WA-OPShare needs to assess which tenant benefits most from them. However, this is quite challenging because the performance benefit of OPS growth is affected by multiple factors, such as write pattern, allocated OPS size, and read/write ratio, as demonstrated in Section II-B3. WA-OPShare solves it through the proposed *OPS marginal gain* model as shown in (10).

Before understanding the OPS marginal gain model, we first describe the relationship between the bandwidth and the above-mentioned factors, as shown in (3)–(9). For fairness, the maximum number of requests a tenant can complete in each scheduling epoch depends on its initial allocated *Budget* which is proportional to its *weight*, as shown in (3). $\text{Cost}_{\text{host\_}w}$, $\text{Cost}_{\text{host\_}r}$, and $\text{Cost}_{\text{extra}}$ indicate the consumed budget of completing host writes, host reads and extra operations inside the SSD (e.g., GC operations), respectively, and they are calculated by (4)–(6). N_{host} is the completed page number of host reads/writes, $N_{\text{extra\_}w}$ and $N_{\text{extra\_}r}$ are the extra writes and reads generated inside the SSD, N_{erase} is the completed block erasing number, t_w, t_r, and t_e are the page write, page read, and block erasing cost, and α_w is the write percentage of N_{host}. $N_{\text{extra\_}w}$ is related to the WAF of the SSD, as shown in (7). We mainly consider the extra operation generated by the GC process in this article. The migration of each valid page in the GC process will generate both an extra read and an extra write, thus the number of extra reads is equal to the number

of extra writes. In a steady-state SSD, the number of free flash pages released by erasing superblocks is equal to the number of flash pages written, as shown in (8), where a flash block consists of P flash pages. Finally, we can get the relationship between the bandwidth and related factors, as shown in (9), by taking (4)–(8) into (3)

$$\text{Cost}_{\text{host\_}w} + \text{Cost}_{\text{host\_}r} + \text{Cost}_{\text{extra}} = \text{Budget} \propto \text{weight} \quad (3)$$

$$\text{Cost}_{\text{host\_}w} = N_{\text{host}} \times \alpha_w \times t_w \quad (4)$$

$$\text{Cost}_{\text{host\_}r} = N_{\text{host}} \times (1 - \alpha_w) \times t_r \quad (5)$$

$$\text{Cost}_{\text{extra}} = N_{\text{extra\_}w} \times t_w + N_{\text{extra\_}r} \times t_r + N_{\text{erase}} \times t_e \quad (6)$$

$$N_{\text{extra\_}w} = N_{\text{host}} \times \alpha_w \times (\text{WAF} - 1) \quad (7)$$

$$N_{\text{erase}} = \frac{N_{\text{host}} \times \alpha_w \times \text{WAF}}{P} \quad (8)$$

$$\text{Bandwidth} = N_{\text{host}} \times \text{PageSize} \propto \frac{\text{weight}}{\alpha_w \times \left[\text{WAF} \times \left(t_r + t_w + \frac{t_e}{P}\right) - 2t_r\right] + t_r} \quad (9)$$

$$\text{MG} = \text{Bandwidth}(\text{WAF} - \Delta x) - \text{Bandwidth}(\text{WAF}). \quad (10)$$

Although the bandwidth is related to several factors, such as tenant's weight, read/write ratio, and WAF of workload, OPS resources improve bandwidth by only decreasing the WAF. Thus, WA-OPShare proposes the *OPS marginal gain* model shown in (10) to predict the marginal gain (MG) of the OPS growth, where Δx indicates the decrease of WAF brought by increasing in unit OPS resource (i.e., a superblock).

To realize an efficient OPS allocation, the OPS allocator module alternates between the *measuring window* that measures Δx for each tenant and the *reallocating window* that allocates the released free superblock from the lazy superblock compaction module to the tenant with the most MG.

1) Measuring Window: It is easy to measure Δx in an offline environment, however, WA-OPShare workload-adaptively adjusts OPS allocation, which needs to measure Δx at runtime. How to predict the benefit Δx of the OPS growth at runtime is challenging. Since the lazy superblock compaction module prefers to release those long-lived invalid pages to increase the available space, which has a similar effect as increasing OPS resources for more available space, WA-OPShare approximately simulates a state of increasing OPS resources.

At the beginning of the measuring window, the current WAF value ($\text{WAF}_{\text{begin}}$) is recorded for each tenant. During the measuring window, the OPS allocator module returns the free superblock released from the lazy superblock compaction module to its original tenant, and records the allocated superblock number (m) of each tenant. At the end of the measuring window, the OPS allocator module records the current WAF value (WAF_{end}), and calculates the decreased WAF value (Δx) of each tenant by (11). We approximately regard this Δx as the reduction in WAF caused by the OPS growth, and its effectiveness has been verified in our evaluations

$$\Delta x = \frac{\text{WAF}_{\text{begin}} - \text{WAF}_{\text{end}}}{m}. \quad (11)$$

During the measuring window, local compaction is performed for each tenant to detect the demand for OPS resources in an independent and fair manner. The superblock with the lowest *efficiency* in each tenant's private superblock pool is reclaimed and returned to the original pool. Meanwhile, the marginal gain is calculated for each tenant.

2) Reallocating Window: The OPS allocator module predicts the marginal gain of the OPS growth by (10) based on each tenant's Δx measured in the measuring window. Whenever the lazy superblock compaction module releases a free superblock in the reallocating window, the OPS allocator module allocates this released superblock to the tenant with the most MG to maximize the benefit of OPS resources. However, to guarantee the performance isolation of tenants whose *Iso* is set as 1, the released superblock from these tenants will be returned to their private superblock pools instead of being allocated to other tenants.

During the reallocating window, global compaction is conducted to reallocate OPS resources between tenants for maximizing the performance gain. Superblocks with the lowest *efficiency* across all tenants are compacted, where free superblocks are released and then reallocated to the tenant whose marginal gain MG is the largest. When a new tenant is admitted to the SSD, the WA-OPShare scheme recycles I/O budget and OPS resources from existing tenants and allocates them to the new tenant until it obtains an initial percentage of resources (the percentage is calculated according to the tenants' weights). The measuring of tenants' WAF and MG values is disabled during this admitting process. Afterward, the measuring restarts and workload-adaptive OPS reallocation between tenants is enabled.

In conclusion, WA-OPShare identifies lazy superblocks by the space efficiency model, monitors the benefit of the OPS growth for each tenant by the OPS marginal gain model, releases the underutilized space of lazy superblocks, dynamically and workload-adaptively allocates it to the tenant with the most marginal gain, and finally achieves the better SSD performance.

IV. Evaluation

A. Experimental Setup

We implement the proposed *WA-OPShare* scheme on the widely used trace-driven SSDsim simulation platform [24]. Table I describes the workload characteristics that we collect by the *blktrace* tool of the Linux system, therein, the varmail, fileserver, and webproxy workloads are provided by the *filebench* tool [25], and the "aerospike" represents the workload of the distributed NoSQL database Aerospike [26]. Table II shows the default SSD configurations [23]. The logical/user-visible capacity and total OPS size are 32 and 8 GB, respectively. The OPS ratio of modern SSDs usually ranges between 7% and 40%, where 25% is a typical and reasonable configuration [27]. In the current implementation, we empirically set the maximum consumed bandwidth by the lazy compaction module as 1% of SSD bandwidth, and measuring window and reallocating window duration in the OPS allocator module as 60 s. In the following evaluations, if not specified, each tenant sharing the multitenant SSD is allocated 16 GB of logical capacity and proportional OPS resources based on its weight by default.

To demonstrate the effectiveness of our WA-OPShare, we evaluate it against two traditional allocation schemes: 1) the Sharing scheme, where superblocks are freely competed by tenants and 2) the Partition scheme, where superblocks are dedicated to separate tenants. Both the Sharing and Partition schemes are implemented upon the software isolated FlashBlox [1] scheme. FlashBlox is a physical isolation scheme for the multitenant environment and can reduce the

TABLE I
WORKLOAD CHARACTERISTICS

Workload	Description	Write Ratio	Avg Req Size (KB)		Accessing Data Set Size (GB)
			Read	Write	
varmail [25]	Many small files with random writes	100%	/	11.4	15.89
fileserver [25]	Many large files with random I/Os	65.4%	43	51.5	15.46
webproxy [25]	Multi-threaded random read, create, write, delete	37.6%	10.2	12.1	13.06
aerospike [26]	Random reads and sequential writes	1.6%	10.1	361.6	14.01

TABLE II
PARAMETERS OF SSDSIM

Parameter	Value	Parameter	Value
flash page read/write	60μs/800μ	channel number	4
flash block erase	1.5ms	chips per channel	4
data transfer rate	400MB/s	dies per chip	1
flash page size	4KB	planes per die	2
flash superblock size	16MB	blocks per plane	1280
OPS	25%	pages per block	256

GC-generated interference among multiple tenants sharing a same SSD. Note that FlashBlox provides both the hardware isolation (the space allocation granularity of channel/die-granularity) and the software isolation (the space allocation of block/superblock-granularity). Here, we evaluate the software-isolated FlashBlox due to its higher parallel bandwidth for each tenant and other advantages mentioned in Section II-A3.

$X(a) - Y(b)$ in evaluations represents that Tenant A runs the workload X with weight a (thus occupies $(a/[a+b])$ of initial budgets and OPS resources) and Tenant B runs the workload Y with weight b. Next we evaluate WA-OPShare from five aspects: 1) the performance improvement of WA-OPShare compared to traditional schemes; 2) the scalability of WA-OPShare; 3) the workload adaptivity of WA-OPShare; 4) the technical decomposition of two key modules of WA-OPShare; and 5) the sensitivity study of WA-OPShare.

B. Performance Comparison

For the traditional Partition and Sharing schemes, initial OPS allocation among tenants has nontrivial impacts on their performance and, thus, requires a careful configuration (which is challenging). In contrast, WA-OPShare can regulate the OPS allocation in an workload-adaptive fashion and always achieve superior performance and WAF, regardless of initial OPS allocation. To verify the effectiveness of WA-OPShare, we study the three schemes' performance under different initial OPS allocation configurations. When the number of tenants is large, the configuration space would be enormous and it would be impractical to verify all the configuration choices. Thus, we consider only two tenants and some typical configurations of initial OPS allocation, i.e., adjusting the allocation between tenants in the unit of 10% of the total OPS size. We found that the Partition scheme obtains the best and worst performance of the "webproxy(1)-fileserver(1)" when allocating workload webproxy 20% and 90% of the OPS resources, respectively, as shown in Fig. 1. The "Fair(50%)" means the proportional OPS allocation based on their weights. We select these three representative OPS allocation configurations for performance comparison, as shown in Fig. 6. For the three schemes evaluated, the "Partition" scheme will not change the OPS allocation during the entire experiment. The "Sharing" scheme allows

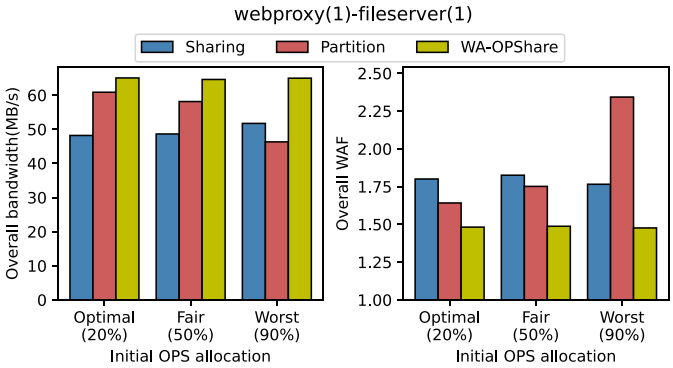

Fig. 6. Comparison of bandwidth and WAF of Sharing, Partition and WA-OPShare schemes with different initial OPS allocation configurations. "webproxy(1)-fileserver(1)" means tenants A and B run the webproxy and fileserver workloads, respectively, with the same weight.

free competition among tenants, causing the OPS allocation among tenants to change until their WAFs are the same. With the goal of enhancing overall performance, the "WA-OPShare" scheme will workload-adaptively adjust OPS allocation among tenants.

As we can find in Fig. 6, compared to the Partition scheme with the "Fair" and "Worst" allocation, WA-OPShare improves bandwidth by 11.1% (from 58.1 to 64.6 MB/s) and 40.3% (from 46.3 to 65.0 MB/s), and decreases WAF by 15.1% (from 1.75 to 1.49) and 37.0% (from 2.34 to 1.48), respectively. Even compared with the "Optimal" allocation, WA-OPShare has a 6.9% improvement of bandwidth (from 60.9 to 65.1 MB/s) and a 9.8% reduction of WAF (from 1.64 to 1.48) (a lower WAF means less write wear and longer lifetime of the SSD). Although the Sharing scheme also changes the initial OPS allocation, compared to it, WA-OPShare achieves approximately 31.2% higher bandwidth and 17.5% lower WAF.

The Partition scheme is vulnerable to the dynamics and changes of workload characteristics, as its performance highly depends on the predefined and static OPS allocation between tenants. It is quite hard to realize optimal OPS allocation, since the workload characteristics of tenants are not known in advance and may be complex and change over time. With improper OPS allocation, the Partition scheme would perform poorly. In contrast, the proposed WA-OPShare scheme is adaptive to the workloads and maximizes the performance gain from OPS resources (given any tenants). Particularly, WA-OPShare can achieve a better performance than the Partition scheme with optimal static OPS allocation.

C. Scalability of WA-OPShare

To demonstrate the scalability of the WA-OPShare scheme in the environment with more concurrent tenants, we expand block numbers per plane in Table II to 2560/3840/5120 to provide a larger capacity for supporting four/six/eight tenants

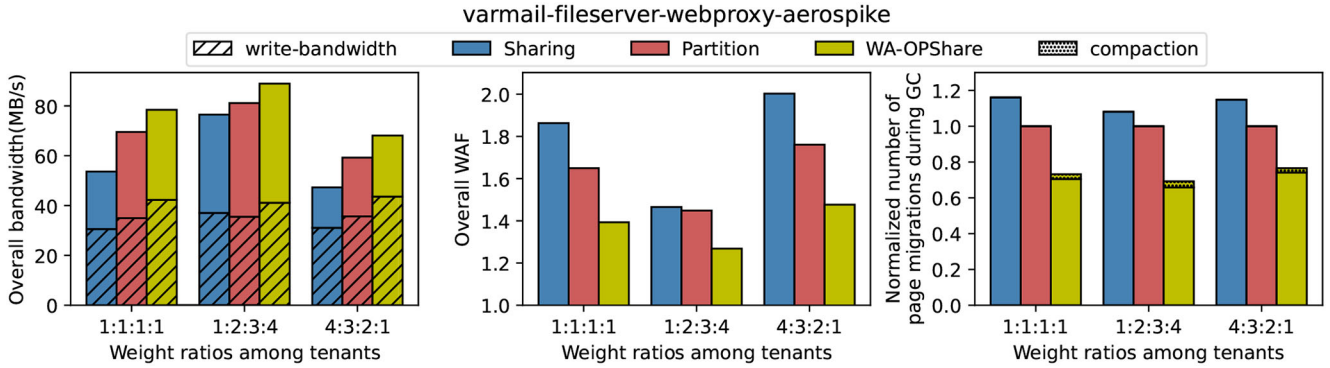

Fig. 7. Bandwidth, WAF, and GC overhead of the Sharing, Partition, and WA-OPShare schemes with different weight ratios among tenants.

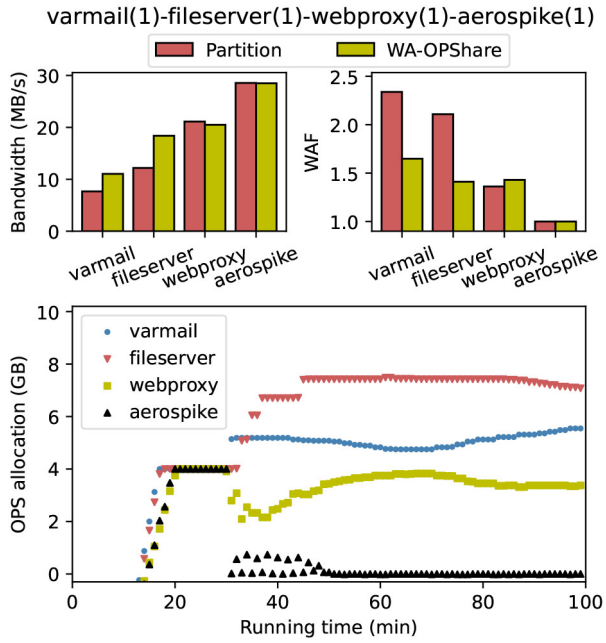

Fig. 8. Comparison of bandwidth and WAF of Partition and WA-OPShare schemes for each tenant and the OPS allocation adjustment of the WA-OPShare scheme.

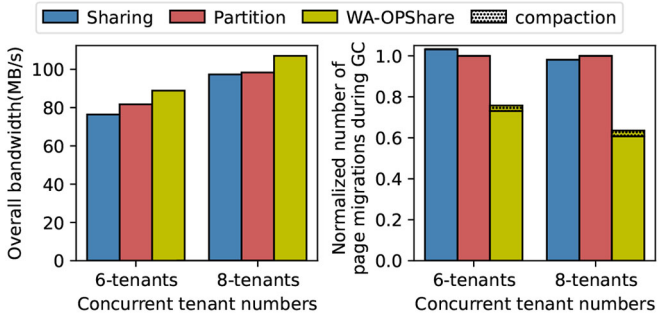

Fig. 9. Comparison of bandwidth and WAF of Sharing, Partition and WA-OPShare schemes when six and eight tenants running concurrently. The "6-tenants" runs extra two random 4-kB I/O workloads "randread" and "randwrite" besides four former workloads, and the "8-tenants" runs extra two sequential workloads "seqread" and "seqwrite" besides the 6-tenants. All tenants have the same weight.

working concurrently. Since there are too many possible OPS allocation configurations among more than two tenants, it is too time consuming to derive the optimal/worst case configuration through brute-force offline experiments; thus, we

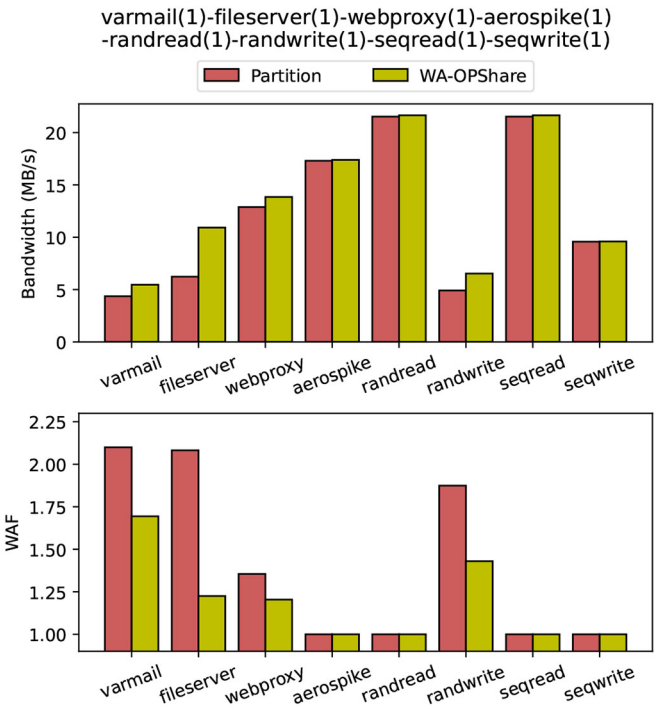

Fig. 10. Comparison of bandwidth and WAF of Partition and WA-OPShare schemes for each tenant when eight tenants running concurrently.

only demonstrate the performance comparison of the Fair OPS allocation configuration.

When four tenants run concurrently, as shown in Fig. 7, we can find that performance of the Sharing scheme is worst due to its arbitrary and ineffective OPS allocation strategy that allows free competition among tenants. Compared to the Partition scheme, our WA-OPShare improves overall bandwidth by 12.8%, 9.4%, and 15.0% in experiments of three different weight ratios, respectively. Since OPS resources mainly help write performance instead of read performance, we focus on the improvement of write performance. Specifically, in the experiment of the weight ratio of 4:3:2:1 among tenants, WA-OPShare improves write bandwidth by 22.3%, decreases WAF by 16.2%, and reduces GC overhead by 23.4%. The experiments of the other two weight ratios ("1:1:1:1" and "1:2:3:4") have similar results that improve write bandwidth by 20.9% and 16.3%, decrease WAF by 15.5% and 12.6%, reduce GC overhead of migrating valid pages by 26.8% and 30.8%, respectively. Since the WA-OPShare scheme limits the

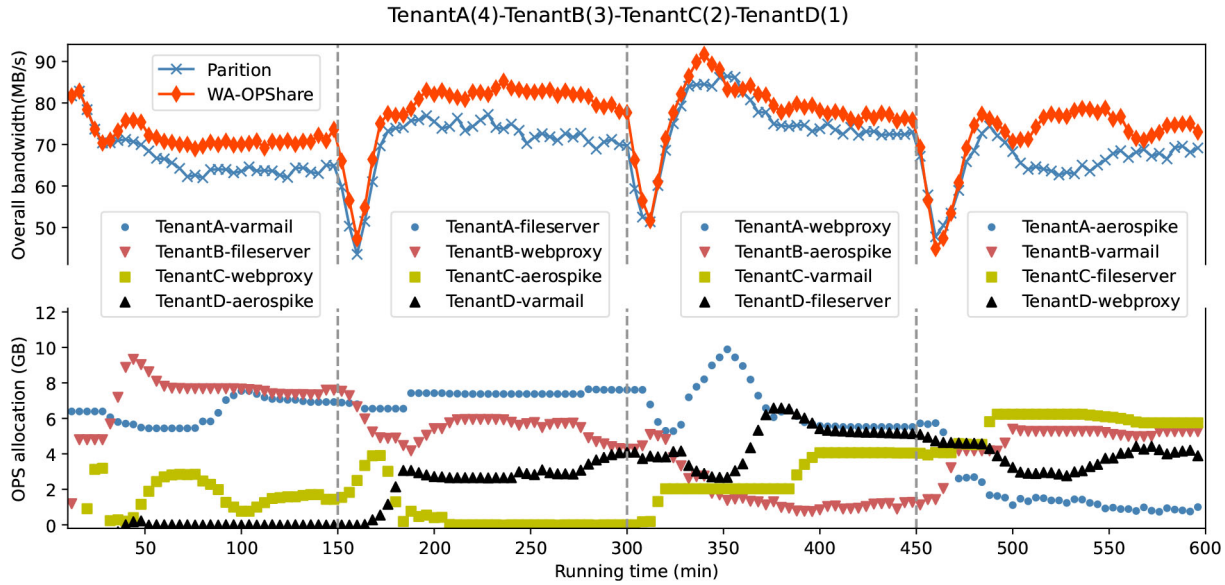

Fig. 11. Overall bandwidth change of the Partition and WA-OPShare schemes at runtime, and the dynamic OPS allocation of the WA-OPShare scheme among tenants. The weight ratio is 4:3:2:1 among four concurrent tenants A, B, C, and D which run varmail, fileserver, webproxy, and aerospike workloads, respectively, at the beginning. The running workloads of tenants change per 150 min.

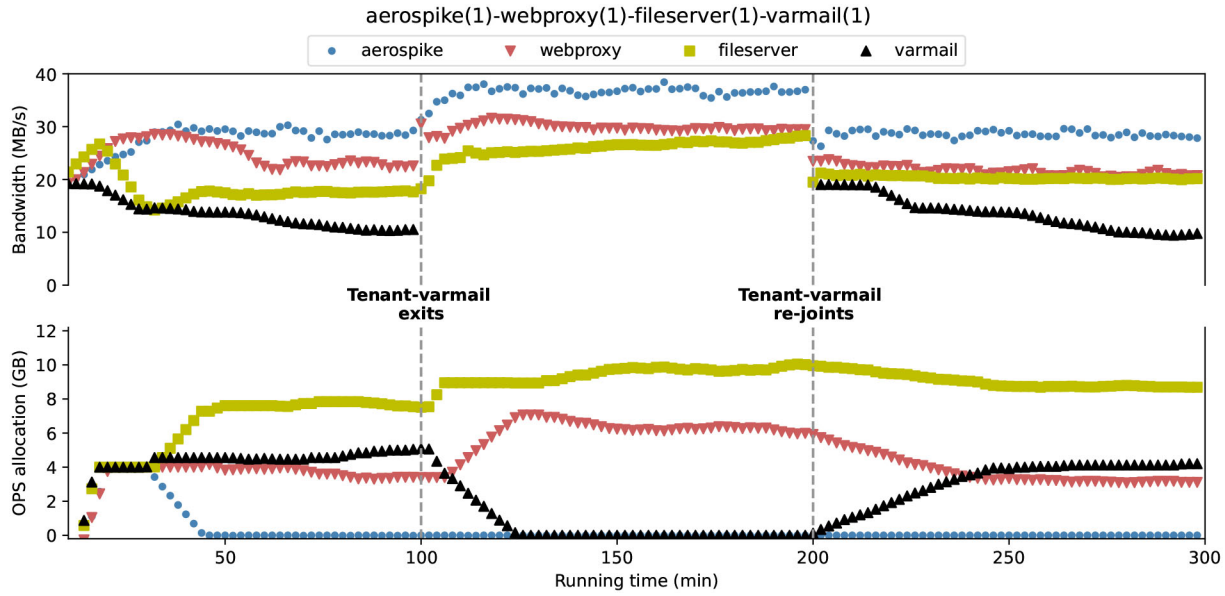

Fig. 12. Bandwidth and OPS allocation change of WA-OPShare at runtime. Tenant varmail exits at 100th min and rejoins at 200th min.

bandwidth consumed for lazy superblock compactions within 1% of the total bandwidth, in the three sets of experiments, page migrations caused by compactions account for only 2.7%, 3.4%, and 2.5% of total page migrations.

The aerospike and webproxy are read-intensive workloads and require few OPS resources, while the varmail and fileserver are write-intensive workloads and require lots of OPS resources. WA-OPShare improves overall performance by dynamically allocating excess OPS resources of read-intensive workloads to write-intensive workloads, as shown in Fig. 8. In the beginning, as workloads run, they begin to gradually preempt OPS resources after consuming their capacities until reaching their initial allocated 4 GB of OPS. Starting from the 30th min, WA-OPShare dynamically adjusts their OPS allocation. The aerospike and webproxy workloads decrease their OPS resources, while the varmail and fileserver workloads increase their OPS resources. Since the aerospike is a read-intensive workload and its write I/Os are all sequential writes, it does not require OPS resources (its WAF is 1), thus decreasing its OPS size to 0. Finally, the varmail has a 43.9% improvement of bandwidth and a 29.5% reduction of WAF, and the fileserver has a 50.9% improvement of bandwidth and a 33.2% reduction of WAF.

When the number of tenants increases to 6 and 8, as shown in Fig. 9, the WA-OPShare scheme improves the overall SSD bandwidth by 8.8% and 8.9% and write bandwidth by 18.4% and 19.2% (without decreasing read bandwidth), and reduces the WAF by 9.1% and 10.1% and extra writes caused by GC by 24.2% and 36.5%, respectively, compared to the Partition scheme. Specifically, in the case of eight tenants sharing an SSD, as shown in Fig. 10, the bandwidth of the three

write-intensive tenants (varmail, fileserver, and randwrite) increases by 25.0%, 75.4%, and 33.0%, respectively.

Furthermore, it is important to note that the workload-adaptivity of the WA-OPShare scheme becomes more significant, when the number of tenants increases. Since the configuration space exponentially grows, it becomes much more challenging for the Partition scheme to make a proper OPS allocation choice. As a conclusion, the WA-OPShare scheme has good scalability on the number of tenants.

In addition, the maximum number of tenants that an SSD can accommodate is restricted by the hardware. For several reliability concerns, each active or partially written flash block/superblock in an SSD requires a set of hardware resources, such as SRAM/DRAM space, power capacitors, and XOR engines. Therefore, an SSD only supports a limited number of active superblocks for concurrent writes. Generally, SSDs allow 4–16 write streams [28] and ZNS SSDs are expected to have 8–32 active zones [29] (each stream or active zone is usually mapped to a superblock). Meanwhile, each tenant requires at least one and typically multiple active superblocks (for hot and cold data separation). As a result, the maximum number of tenants sharing an SSD should be quite limited. We do not extend the experiments with a larger number of tenants than 8.

D. Workload Adaptivity of WA-OPShare

To demonstrate the adaptivity of WA-OPShare to workload variability, we change the running workloads of four concurrent tenants per 150 min, as shown in Fig. 11. Thanks to the OPS marginal gain model, the change in workload characteristics can be monitored in time. WA-OPShare can release excess OPS resources of the tenant whose running workload changes from write intensive to read intensive and reallocate these released OPS resources to the OPS-starved workloads to improve SSD performance. Specifically, as shown in Fig. 11, whenever the running workloads of tenants change (at 150th, 300th, and 450th min), WA-OPShare can always dynamically allocate the OPS resources of OPS-abundant tenants running the read-intensive workloads (webproxy and aerospike) to OPS-starved tenants running the write-intensive workloads (varmail and fileserver); therefore, WA-OPShare improves the SSD bandwidth by an average of 11%, 14%, 7%, and 25% in the four stages, respectively. These results verify the workload adaptivity of the WA-OPShare scheme and also the vulnerability of the Parition scheme.

We also demonstrate the impacts of the exiting and joining of a tenant on the performance and OPS allocation. As shown in Fig. 12, after a tenant exits in the 100th min and rejoins in the 200th min, OPS resources are gradually reallocated between existing tenants. During the departure of the tenant, the remaining tenants obtain more I/O budget and OPS resources and, thus, their bandwidth increases. At the beginning of the tenant rejoining the SSD, the WA-OPShare scheme recycles I/O budget and OPS resources from existing tenants and allocates them to the new tenant. The measuring of tenants' WAF and MG values is disabled until the rejoined tenant obtains an initial percentage of resources according to its weight. Afterward, the measuring restarts and workload-adaptive OPS reallocation between tenants is enabled.

E. Technical Decomposition of WA-OPShare

To provide deep insights about WA-OPShare, we also implement two more intermediate schemes: 1) *Partition + Compaction* and 2) *Sharing + Compaction*, which, respectively, integrates only the lazy superblock compaction module of the WA-OPShare scheme into the traditional Partition and Sharing schemes, and their performance is shown in Fig. 13.

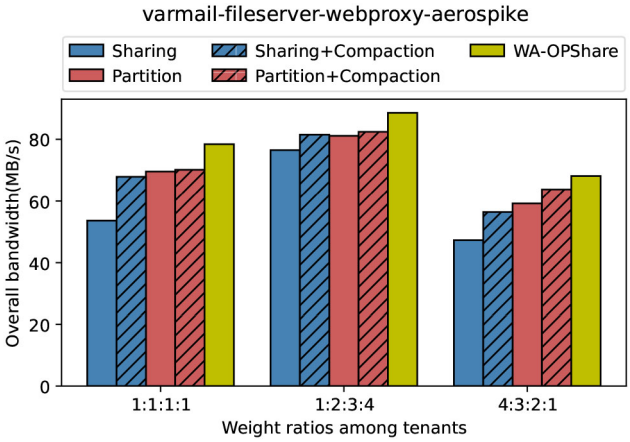

Fig. 13. Performance comparison of five schemes in the experiments of running varmail, fileserver, webproxy, and aerospike workloads with different weight ratios.

We can find that the lazy superblock compaction works well when it is integrated in the Sharing scheme. By compacting lazy superblock to release the space of long-lived invalid pages, the "Sharing + Compaction" scheme improves the utilization efficiency of OPS resources and achieves a 26.5%, 9.8%, and 19.3% improvement of bandwidth in three different weight ratios, respectively. Meanwhile, the "Partition + Compaction" scheme only works in the experiment of the weight ratio of "4:3:2:1" with a bandwidth improvement of 7.5% but does not work in the other two experiments. We analyze that the reason is that the weights of write-intensive workloads (varmail and fileserver) that can benefit from the released OPS resources are high in the experiment of weight ratio of 4:3:2:1. Nevertheless, these two intermediate schemes integrating only the lazy superblock compaction module are not taking full advantage of OPS resources. By allocating these released space to the tenant who can benefit most from it, WA-OPShare can always achieve an better performance.

F. Sensitivity Study of WA-OPShare

In this section, we perform sensitivity studies on the OPS ratio configuration and design parameters in WA-OPShare. WA-OPShare improves the performance of a multitenant SSD by reallocating OPS resources between tenants in a workload-adaptive manner. As shown in Fig. 14, WA-OPShare always achieves better performance than the Partition and Sharing schemes under a range of OPS ratio configurations, i.e., 5% to 40%. When the OPS ratio is set very low, tenants are short of OPS resources and the room for dynamic OPS reallocation between tenants is limited. Thus, the bandwidth improvement of our proposed WA-OPShare scheme over the Partition scheme is small, e.g., 5.1% under the 5% OPS ratio configuration. As the OPS ratio increases, the WA-OPShare scheme becomes more effective, such as improving the overall bandwidth by 10.3% and write bandwidth by 14.7% under the 20% OPS ratio configuration. Then, when the OPS ratio becomes high, the benefits diminish because tenants are less demanding for OPS resources, for example, the bandwidth improvement is 9.3% under the 40% OPS ratio configuration.

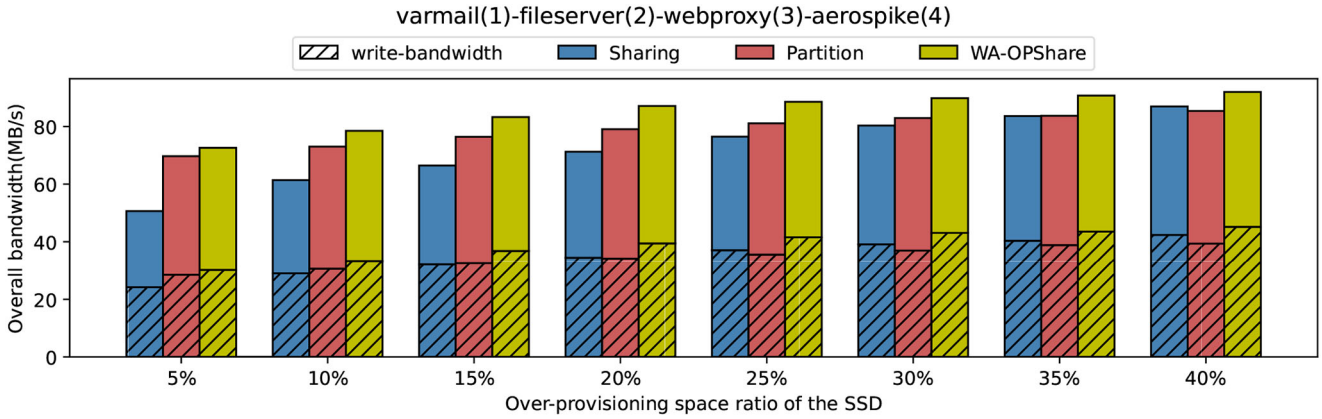

Fig. 14. Comparison of bandwidth of Sharing, Partition, and WA-OPShare schemes with different OPS configurations of the SSD.

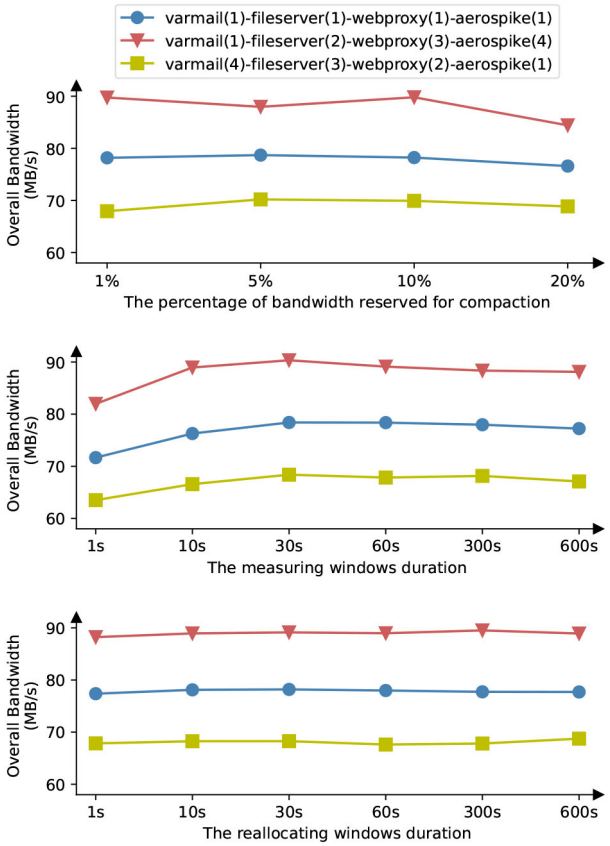

Fig. 15. Performance of different parameter configurations of WA-OPShare.

Although the compaction of lazy superblocks consumes SSD bandwidth, the released underutilized space also reduces GC overhead, which offsets its compaction overhead. WA-OPShare monitors the marginal gain of the OPS growth for tenants in each measuring window, and dynamically adjusts the OPS allocation among tenants in each reallocating window. A shorter measuring window duration reduces measurement accuracy and a shorter reallocating window duration slows down the adjustment of dynamic OPS allocation, while a longer measuring window and reallocating window duration increases hysteresis of measurement. We set the reserved bandwidth for the lazy compaction as 1% of SSD bandwidth, and measuring window and reallocating window duration as 60 s by default, and studied the impact on performance when one parameter changes, as shown in Fig. 15. We can see the performance of WA-OPShare is not sensitive to these parameters. Most configuration results in relatively good performance except for the too-short measuring window duration (1 s). Due to the too-short duration of the measurement, the measurement results are inaccurate and, thus, cannot effectively guide the dynamic allocation of OPS resources. Therefore, we moderately set the reserving of 1% and window duration of the 60 s in our implementation.

V. Conclusion

In this article, we showed that traditional space Partition and Sharing schemes cannot fully utilize OPS resources and are both suboptimal-performing. To improve the utilization of OPS resources, we proposed WA-OPShare, a workload-adaptive OPS allocation scheme for the multitenant SSD. Its space efficiency model can identify lazy superblocks and release long-lived invalid space regularly for better utilizing OPS resources. Moreover, its OPS marginal gain model can monitor the OPS benefit of the OPS growth for each tenant, and advise a wise OPS allocation for preventing tenants from holding too many OPS than needed which results in low efficiency. We have conducted extensive experiments to verify WA-OPShare delivers an effective solution to improve the overall performance of the multitenant SSD. As future work, we plan to borrow wisdom from other fields (e.g., machine learning [30], [31]) to improve the detection and utilization of I/O patterns for multitenant storage management.

References

[1] J. Huang *et al.*, "FlashBlox: Achieving both performance isolation and uniform lifetime for virtualized SSDs," in *Proc. FAST*, 2017, pp. 375–390.
[2] T. Luo, S. Ma, R. Lee, X. Zhang, D. Liu, and L. Zhou, "S-CAVE: Effective SSD caching to improve virtual machine storage performance," in *Proc. PACT*, 2013, pp. 103–112.
[3] F. Meng, L. Zhou, X. Ma, S. Uttamchandani, and D. Liu, "vCacheShare: Automated server flash cache space management in a virtualization environment," in *Proc. USENIX Annu. Tech. Conf.*, 2014, pp. 133–144.
[4] R. Birke, M. Björkqvist, L. Y. Chen, E. Smirni, and T. Engbersen, "(Big)data in a virtualized world: Volume, velocity, and variety in cloud datacenters," in *Proc. FAST*, 2014, pp. 177–189.
[5] B. S. Kim, "Utilitarian performance isolation in shared SSDs," in *Proc. HotStorage*, 2018, p. 16.
[6] A. Tavakkol *et al.*, "FLIN: Enabling fairness and enhancing performance in modern NVMe solid state drives," in *Proc. ISCA*, 2018, pp. 397–410.

[7] P. Goyal, H. M. Vin, and H. Cheng, "Start-time fair queueing: A scheduling algorithm for integrated services packet switching networks," *IEEE/ACM Trans. Netw.*, vol. 5, no. 5, pp. 690–704, Oct. 1997.
[8] K. Shen and S. Park, "FlashFQ: A fair queueing I/O scheduler for flash-based SSDs," in *Proc. USENIX Annu. Tech. Conf.*, 2013, pp. 67–78.
[9] M. Hedayati, K. Shen, M. L. Scott, and M. Marty, "Multi-queue fair queuing," in *Proc. USENIX Annu. Tech. Conf.*, 2019, pp. 301–314.
[10] X.-Y. Hu, E. Eleftheriou, R. Haas, I. Iliadis, and R. Pletka, "Write amplification analysis in flash-based solid state drives," in *Proc. SYSTOR*, 2009, p. 10.
[11] S. Park and K. Shen, "FIOS: A fair, efficient flash I/O scheduler," in *Proc. FAST*, 2012, p. 13.
[12] B. Jun and D. Shin, "Workload-aware budget compensation scheduling for NVMe solid state drives," in *Proc. NVMSA*, 2015, pp. 1–6.
[13] D.-W. Chang, H.-H. Chen, and W.-J. Su, "VSSD: Performance isolation in a solid-state drive," *ACM Trans. Des. Autom. Electr. Syst.*, vol. 20, no. 4, pp. 1–33, 2015.
[14] J. Liu, F. Wang, and D. Feng, "CostPI: Cost-effective performance isolation for shared NVMe SSDs," in *Proc. ICPP*, 2019, pp. 1–10.
[15] "Single root I/O virtualization." Accessed: Apr. 7, 2022. [Online]. Available: https://www.pcisig.com/specifications/iov/single$_{r}$oot/
[16] "NVM express 1.3 specification." 2019. [Online]. Available: https://nvmexpress.org/
[17] Y. Cai, S. Ghose, E. F. Haratsch, Y. Luo, and O. Mutlu, "Error characterization, mitigation, and recovery in flash-memory-based solid-state drives," *Proc. IEEE*, vol. 105, no. 9, pp. 1666–1704, Sep. 2017.
[18] Y. Zhou, Q. Wu, F. Wu, H. Jiang, J. Zhou, and C. Xie, "Remap-SSD: Safely and efficiently exploiting SSD address remapping to eliminate duplicate writes," in *Proc. FAST*, 2021, pp. 187–202.
[19] Q. Zhang, D. Feng, F. Wang, and Y. Xie, "An efficient, QoS-aware I/O scheduler for solid state drive," in *Proc. HPCC/EUC*, 2013, pp. 1408–1415.
[20] A. J. Demers, S. Keshav, and S. Shenker, "Analysis and simulation of a fair queueing algorithm," in *Proc. ACM SIGCOMM*, 1989, pp. 1–12.
[21] X. Song, J. Yang, and H. Chen, "Architecting flash-based solid-state drive for high-performance I/O virtualization," *IEEE Comput. Archit. Lett.*, vol. 13, no. 2, pp. 61–64, Jul.–Dec. 2014.
[22] Y. Zhou, F. Wu, W. Huang, and C. Xie, "LiveSSD: A low-interference RAID scheme for hardware virtualized SSDs," *IEEE Trans. Comput.-Aided Design Integr. Circuits Syst.*, vol. 40, no. 7, pp. 1354–1366, Jul. 2021.
[23] J. Kim, D. Lee, and S. H. Noh, "Towards SLO complying SSDs through OPS isolation," in *Proc. FAST*, 2015, pp. 183–189.
[24] Y. Hu, H. Jiang, D. Feng, L. Tian, H. Luo, and C. Ren, "Exploring and exploiting the multilevel parallelism inside SSDs for improved performance and endurance," *IEEE Trans. Comput.*, vol. 62, no. 6, pp. 1141–1155, Jun. 2013.
[25] "Filebench." Accessed: Apr. 7, 2022. [Online]. Available: https://github.com/filebench/filebench/wiki
[26] "Aerospike." Accessed: Apr. 7, 2022. [Online]. Available: https://github.com/aerospike/aerospike-server
[27] J. Ouyang, S. Lin, S. Jiang, Z. Hou, Y. Wang, and Y. Wang, "SDF: Software-defined flash for Web-scale Internet storage systems," in *Proc. ASPLOS*, 2014, pp. 471–484.
[28] T. Kim *et al.*, "Fully automatic stream management for multi-streamed SSDs using program contexts," in *Proc. FAST*, 2019, pp. 295–308.
[29] M. Bjørling *et al.*, "ZNS: Avoiding the block interface tax for flash-based SSDs," in *Proc. USENIX Annu. Tech. Conf.*, 2021, pp. 689–703.
[30] E. Deniz and A. Sen, "Using machine learning techniques to detect parallel patterns of multi-threaded applications," *Int. J. Parallel Program.*, vol. 44, no. 4, pp. 867–900, 2016.
[31] M. Hao, L. Toksoz, N. Li, E. E. Halim, H. Hoffmann, and H. S. Gunawi, "LinnOS: Predictability on unpredictable flash storage with a light neural network," in *Proc. OSDI*, 2020, pp. 173–190.

Yuhong Wen received the B.E. degree in computer science and technology from the Huazhong University of Science and Technology, Wuhan, China, in 2020, where he is currently pursuing the Ph.D. degree in computer architecture with the Wuhan National Laboratory for Optoelectronics.

His research interests include nonvolatile memory, solid-state storage architectures and systems, and multitenant storage quality of service.

You Zhou received the B.E. degree in computer science and technology and the Ph.D. degree in computer architecture from the Huazhong University of Science and Technology (HUST), Wuhan, China, in 2011 and 2017, respectively.

He worked as a Postdoctoral Researcher with the Wuhan National Laboratory for Optoelectronics, HUST, from 2018 to 2020, where he is currently an Associate Professor with the School of Computer Science and Technology. His research interests include nonvolatile memory, solid-state storage architectures and systems, storage quality of service, and hardware–software co-designs.

Fei Wu (Member, IEEE) received the B.E. and M.E. degrees in electrical automation, control theory, and control engineering from Wuhan Industrial University, Wuhan, China, in 1997 and 2000, respectively, and the Ph.D. degree in computer science from the Huazhong University of Science and Technology (HUST), Wuhan, in 2005.

She is currently a Professor with the Wuhan National Laboratory for Optoelectronics, HUST. Her research interests include computer architecture, nonvolatile memory, and intelligent storage.

Shu Li received the bachelor's and master's degrees from the Department of Electrical Engineering, Tsinghua University, Beijing, China, in 2003 and 2005, respectively, and the Ph.D. degree from the Department of Electrical Engineering, Rensselaer Polytechnic Institute, Troy, NY, USA, in 2009.

He is currently serving as the Chief Storage Scientist for Alibaba Cloud, Hangzhou, China, and leads the Alibaba storage infrastructure research, development, deployment, and operation.

Zhenghong Wang received the bachelor's degree in radio and television engineering from the Nanjing University of Posts and Telecommunications, Nanjing, China, in 2008, and the master's degree in computer software and theory from the Chongqing University of Posts and Telecommunications, Chongqing, China, in 2011.

He is currently serving as the Senior Expert for Alibaba Cloud, Hangzhou, China, and leads the Alibaba self-developed high-performance storage server, software, and hardware.

Changsheng Xie received the B.E. and M.E. degrees in computer science and technology from the Huazhong University of Science and Technology (HUST), Wuhan, China, in 1982 and 1988, respectively.

He is currently a Professor with the Wuhan National Laboratory for Optoelectronics, HUST. He is also the Director of the Data Storage Systems Laboratory, HUST, and the Key Laboratory of Ministry of Education of China. His research interests include computer systems and networks and emerging storage technologies.