Fair Write Attribution and Allocation for Consolidated Flash Cache

Wonil Choi
Pennsylvania State University
wuc138@cse.psu.edu

Bhuvan Urgaonkar
Pennsylvania State University
bhuvan@cse.psu.edu

Mahmut Kandemir
Pennsylvania State University
mtk2@cse.psu.edu

Myoungsoo Jung
KAIST
mj@camelab.org

David Evans
Samsung Semiconductor
david.evans@samsung.com

Abstract

Consolidating multiple workloads on a single flash-based storage device is now a common practice. We identify a new problem related to *lifetime management* in such settings: how should one partition device resources among consolidated workloads such that their allowed contributions to the device's wear (resulting from their writes including hidden writes due to garbage collection) may be deemed *fairly assigned*? When flash is used as a cache/buffer, such fairness is important because it impacts what and how much traffic from various workloads may be serviced using flash which in turn affects their performance. We first clarify why the write attribution problem (i.e., which workload contributed how many writes) is non-trivial. We then present a technique for it inspired by the *Shapley value*, a classical concept from cooperative game theory, and demonstrate that it is accurate, fair, and feasible. We next consider how to treat an overall "write budget" (i.e., total allowable writes during a given time period) for the device as a first-class resource worthy of explicit management. Towards this, we propose a novel write budget allocation technique. Finally, we construct a dynamic lifetime management framework for consolidated devices by putting the above elements together. Our experiments using real-world workloads demonstrate that our write allocation and attribution techniques lead to performance fairness across consolidated workloads.

CCS Concepts. • **Information systems** → **Flash memory**; **Storage management**; • **Theory of computation** → **Solution concepts in game theory**.

Keywords. flash lifetime, fair allocation, Shapley value

ASPLOS '20, March 16–20, 2020, Lausanne, Switzerland

ACM ISBN 978-1-4503-7102-5/20/03…$15.00
https://doi.org/10.1145/3373376.3378502

ACM Reference Format:
Wonil Choi, Bhuvan Urgaonkar, Mahmut Kandemir, Myoungsoo Jung, and David Evans. 2020. Fair Write Attribution and Allocation for Consolidated Flash Cache. In *Proceedings of the Twenty-Fifth International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS '20), March 16–20, 2020, Lausanne, Switzerland.* ACM, New York, NY, USA, 14 pages. https://doi.org/10.1145/3373376.3378502

1 Introduction

With growing capacity and penetration of flash devices, it is increasingly common to see multiple workloads consolidated within a single device, be it a single solid-state drive (SSD) or a flash array [11, 12, 50, 66–68]. Compared to hosting just one workload, as has been the common case in the past, the emerging consolidated devices are expected to face a significantly increased intensity of I/O requests and/or storage capacity demands to accommodate multiple workloads' data.

It is inevitable in any system provisioned in a cost-conscious manner (i.e., for less than anticipated worst-case needs) that sometimes the needs of users ("demand") exceed available capacity ("supply"). Episodes of such supply-demand mismatch will only increase in their severity (frequency and duration) in consolidated SSDs. Two canonical concepts (both accompanied by "demand shedding," e.g., by admission control) have long been used for resource allocation during supply-demand mismatches across a variety of computing systems: priorities (relative importance of workloads) and fairness. When priorities are identical notions of fairness offer principled ways for resource allocation during periods of scarcity.

Lifetime Fairness: What and Why? Multiple SSD resources may need to be fairly partitioned in consolidated settings, among which bandwidth to host (in bytes/sec or I/O operations/sec, IOPS, as appropriate) [19, 20, 27, 28, 49, 55, 58, 60, 61] and storage capacity [2–4, 31, 37, 39] have received the most attention in literature. Flash is commonly used to form a read cache and write buffer layer (simply "cache" henceforth) between the volatile DRAM and the slower persistent magnetic hard disk drives (HDDs). We argue that, in such a flash-based caching layer, an additional resource – **flash lifetime** – should be viewed as a *first-class resource at par with capacity and bandwidth that should be explicitly allocated in principled ways suggested by notions of fairness.*

In some storage systems, e.g., high-end clouds for financial services [16, 47], it may be possible to use the flash device without being constrained by lifetime concerns (i.e., the higher cost of more frequent flash replacement may be acceptable). However, not all settings have this luxury and administrators may want to have more control over how long a flash device lasts. In fact, even in the more high-end environments, during some tail end of a flash device's lifetime (the specifics being device and workload dependent), there may be a need to ensure the device lasts a certain period rather than expiring suddenly. Reasons for being reluctant to replace devices unexpectedly are numerous. For one, the transfer of contents from the about-to-fail device to its replacement may only be possible during idle periods so as not to degrade foreground workload performance. Also, market forces have been known to cause unpredictable price hikes due to shortage of flash devices [42].

Since operating the device for a prescribed duration would be made difficult by the lack of predictability about workload properties (especially too far into the future), some form of limiting writes over relatively short periods of reasonable predictability (we call these "epochs") would be needed. Specifically, in each epoch for which a fixed number of available writes (we call this "overall write budget") is prescribed by the administrator, the budget may be divided among the consolidated workloads and each individual workload may be allowed to issue only the number of writes allocated to it (with the remainder serviced by a slower HDD)[1]. This sets up the context for the problem that we address in this paper: *given an epoch and the overall write budget prescribed to the epoch, how do we fairly allocate the budget across competing workloads running together during the epoch?*

Importantly, we find that, ignoring lifetime (write budget) and only limiting fairness concerns to bandwidth and capacity can result in performance unfairness across consolidated workloads – the difference between response times of consolidated workloads is an intuitively appealing metric for performance fairness; closer the response times, fairer the allocation. Specifically, our experiments reveal that *non-fair* write allocations (not to mention the case of *no* write allocation) result in significant differences in the response times of consolidated workloads. This is because workloads allocated larger portions of the write budget can get performance benefits from using the faster flash layer for relatively more of their write operations.

Why Fair Lifetime Allocation is Challenging: Defining what a fair allocation means for flash lifetime (i.e., the write budget) is much less clear than it is for conventional resources. There are two main reasons for this:

- *Indirect control:* Unlike a resource like bandwidth, a resource manager cannot directly control the "consumption" of lifetime. Flash lifetime consumption occurs due to the servicing of writes by the flash device. The number of writes itself depends on the allocation of over-provisioned (OP) capacity, user-transparent extra storage capacity on a flash device – see Section 2.1 for more details. Besides the writes a workload directly issues (host writes), an additional (often major) contributor of writes is the garbage collection (GC). The number of GC writes a workload causes is affected by the OP capacity allocated to it [13, 53, 59] – in general, the more OP capacity, the fewer the GC writes. Thus, a fair write allocation needs to be realized by a correspondingly suitable allocation of OP capacity.
- *Non-trivial attribution:* A key building block for implementing fair resource allocation is accurate *attribution* of the consumption of the resource to the users. However, write attribution in flash is far from being a simple matter. To appreciate this, suppose a workload A, when using an SSD without any other co-located workloads, results in total of $W(A) = H(A) + G(A)$ writes, where $H(A)$ denotes the number of host writes, $G(A)$ the number of GC writes, and $W(A)$ the total number of writes. When two workloads A and B are co-located on that same SSD, generally, the total number of writes generated can be expressed as $W(A,B) = H(A)+H(B)+G(A,B) > (H(A)+G(A))+(H(B)+G(B))$, where we denote by $G(A,B)$ and $W(A,B)$ the number of GC writes and total writes generated when A and B are co-located – see Table 1 as an example where two scenarios using three real-world workloads are compared in terms of the number of GC writes: separate vs combined. This is because with smaller OP allocation, each workload's GC writes grow. The difference, $G(A,B) - (G(A) + G(B))$, crucially depends on how OP capacity was allocated to A and B. *How should* $G(A,B) - (G(A) + G(B))$ *be attributed to A and B?* It is not clear at all what the "right" answer to this is.

Table 1. Simulated separate vs combined GC writes for real-world workloads. Details of the workloads and experimental setup can be found in Table 4 and Section 7.1. G(·) means the numbers of GC writes. G(W1), G(W2), or G(W3) represents the case where each workload uses the entire OP space, whereas G(W1,W2,W3) indicates the case where workloads W1, W2, and W3 share the OP space. Generally, (G(W1)+G(W2)+G(W3)) < G(W1,W2,W3).

G(W1)	G(W2)	G(W3)	G(W1)+G(W2)+G(W3)	G(W1,W2,W3)
1,602,427	2,813,291	173,492	4,589,210	7,894,349

Our Approach: For the write attribution problem, we draw upon the idea of Shapley value [54] from classical economics. Here the setting is that of a *coalitional game* (wherein the participants play in compliance with the rules of the game set by an external agent) and the problem is that of fairly dividing the proceeds of the game (i.e., total gains) among

[1] Our lifetime management knob is write regulation, which is an option only in systems where writes disallowed by flash can be serviced by some other persistent medium. Examples include flash as a cache/buffer for HDD (which we study) [15, 34, 44, 46, 56] or a secondary store combining SSD and HDD [45, 62, 65]. For purely SSD-based secondary stores, such write regulation is not an option (i.e., as all writes must be serviced by the flash devices) which makes lifetime fairness a moot and ill-posed concern.

the participants. The write budget and portions of the budget attributed to individual workloads are analogous to the total gain and the portions of the total gain assigned to individual players, respectively. The behavior of the flash drive (e.g., how GC writes reduce with an increase in OP capacity) is analogous to the rules set by the external entity.

For the write allocation problem, we first consider *feasible* allocations that meet two conditions: (i) the write allocations for individual workloads do not exceed the total budget and (ii) the corresponding allocations of OP capacity do not exceed the total OP capacity. Specifically, we employ an existing GC estimation model [13] and explore the interplay between the number of GC writes to be generated and the OP capacity to be allocated for each workload. Among such feasible allocations, we then explore multiple intuitively appealing allocation techniques and empirically compare them for their efficacy in offering performance fairness.

Contributions: We make the following main contributions:

- We identify the write allocation problem in consolidation settings: in a certain time duration (epoch), how can one fairly allocate the total number of available writes (budget) to the competing workloads? We also find that a fair budget allocation should be followed by the corresponding OP allocation, a workload's write consumption depends on its allocated OP size. We devise and empirically compare four different write allocation strategies, some of which are inspired by the notions from classical economics max-min fairness and Shapley value.
- We propose a novel write attribution mechanism, inspired by the notion of fairness provided by Shapley value, as a basis for fair write allocation. We also address some difficulties such as (i) estimating the total writes in unobserved subsets and (ii) applying it to large sets. One interesting observation with this mechanism is that a certain fraction of the writes are accounted to read-only workloads as well.
- We propose a novel lifetime management mechanism for a consolidated flash cache, which puts the write allocation (division of total budget and total OP), the write attribution (accounting of consumed writes), and the write control (suppressing write consumption beyond the allocated budget) all together. We implement this mechanism within (resource-rich) host system, as the (computation-intensive) attribution process can be offloaded from flash devices. Using this, device operators can make the consolidated workloads consume the lifetime of their flash device fairly and efficiently.

2 Background and Related Work

2.1 Flash Device Basics

A flash device contains several flash packages as its storage medium and a set of hardware and software modules to manage these packages. Each package consists of thousands of "blocks," where a block is the erase unit for flash memory. The blocks are grouped into dies or planes, which can be accessed in parallel. Each block is divided into hundreds of "pages," where a page is the unit for reads and writes. A key feature of flash devices, especially relevant to our focus on lifetime management, is that they include some *extra* capacity (in the form of flash blocks) *beyond* what is advertised by the vendors. This extra capacity, which is referred to as *over-provisioned* (OP) capacity, is invisible to the user and can be expressed as a fraction of the user-perceived capacity [53, 59]. The main purpose of this OP capacity is to help reduce the performance overheads and lifetime impact of garbage collection, which we discuss next.

Flash devices need a garbage collection (GC) mechanism, since (i) the erase unit is much larger than the write unit and (ii) overwriting data in-place is *not* allowed. When the available "clean space" (i.e., pages that are ready to be written) in a device goes below a threshold, GC is invoked to secure more clean blocks by using an appropriate combination of writes and erases. When invoked, it selects the (to-be-erased) victim blocks, copies the valid page data from them to other blocks, and finally erases them. Thus, the actual number of writes that the device handles is typically *larger* than the number of explicit (host-issued) writes. These GC writes are, therefore, aptly called "amplified" or "hidden" writes. The ratio of total writes to host-issued writes is called the write amplification factor (WAF). Among possible contributors to the amplified writes (e.g., wear-leveling, data-refresh, and parity writes), we mainly focus on GC writes, since they are (i) *unknown* (the number of GC writes a device experiences is complicatedly related to its OP capacity; in general, the more OP capacity, the less GC writes) and (ii) *on demand* (they may not be deferred unlike wear-leveling and data-refresh).

As flash memory services more and more write and erase operations, its cells get damaged progressively till they are no longer usable. The wear-out rates of cells are close to one another as they are grouped into pages and blocks that are used in a "wear-leveled" fashion. Wear leveling refers to mechanisms within the device that try to keep the number of writes/erases to its pages/blocks about the same. Therefore, vendors generally specify the lifetime of their devices in terms of the number of sustainable writes or erases, which are referred to as the "program/erase (P/E) cycles." We use *write count* as the lifetime indicator, where the count includes both host writes and GC writes.

2.2 Our Target System

Our target device is a flash device that forms *a part of a caching + buffering layer* between machines/hosts running our workloads (compute layer) and a slower HDD-based persistent layer (storage layer). This represents a very popular manner in which flash devices are currently used [2–4, 31, 37, 39]. Figure 1(a) illustrates the overview of our target system where the host system employs a storage hierarchy consisting of a flash-tier and a HDD-tier. We assume that the flash device is partially managed by the host system. This is an important flash-related trend both in research and practice (e.g., open-channel SSD [6], software-defined flash [48],

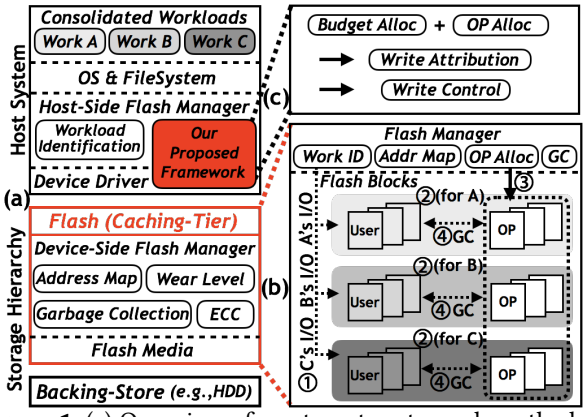


Figure 1. (a) Overview of our target system where the host employs a flash device as caching-tier and a HDD as backing-store; (b) our target flash device is consolidation-oriented, which includes functionalities commonly assumed in literature; (c) our proposed framework is located within the host system, which is independent from flash device type.

and cooperative flash management [23]), due to advantages such as awareness of both host and device information and application-aware flash management [10, 18, 21, 51].

Among possible designs of a flash device on which multiple workloads are consolidated, we target *a soft-partitioned SSD*, as described in Figure 1(b). While detailed functionalities can slightly vary across soft-partitioned SSD designs [11, 27, 28, 50], our target device has the following key characteristics: ① it is *workload-aware*, i.e., can distinguish and treat differently requests belonging to different workloads; ② a flash block is not shared by different workloads at a given time; ③ it employs *soft partitioning* wherein an erased flash block can be allocated to a workload different from the one using it before the erasure; hence, partitioning the total user and OP capacities is realized by dividing the total user and OP blocks among workloads and by allowing each workload to use only a number of flash blocks reflecting its allocation; and ④ it employs a *per-workload GC* mechanism; if needed, each workload is responsible for securing clean blocks by invoking its own GC based on its allocated user and OP blocks.

We highlight the last two functionalities (③ and ④), which are used to explore division of OP blocks among consolidated workloads and the number of GC writes each workload will generate. Specifically, we allocate the total OP capacity (blocks) to consolidated workloads as a part of our lifetime allocation, and each workload is forced to use only its allocated OP blocks for GC. One can implement these functionalities using an open-channel SSD [6] or add necessary interfaces to a multi-stream SSD [11].

2.3 Scope of Our Work

While there exist many variants, we target flash caching-tier which acts as a write buffer as well as a read cache operates as follows. Figure 2 illustrates how each workload uses its allocated blocks and what kinds of traffic it experiences. The

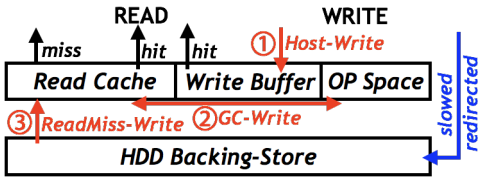


Figure 2. Each workload has its own user blocks (used as both read cache and write buffer) and OP blocks. This flash cache experiences three types of writes: (1) writes from the host, (2) writes during GC, and (3) writes from read misses.

user capacity (blocks) assigned to a workload is divided into two parts: one part works as a read cache that accommodates a set of frequently-read data, and the other part is used as a write buffer that handles all host writes regardless of their target addresses. In this configuration [46], read and write requests get serviced as follows:

• **Read:** If either read cache or write buffer holds (a valid version of) the requested content, the data is read to the host ("read hit"). Otherwise ("read miss"), the data is read from the HDD. We assume a least-recently-used (LRU) cache eviction policy for moving victims from read cache to HDD. However, our framework is agnostic to the exact replacement policy. The data admission (③) leads to a write to flash device.

• **Write:** Writes from the host (①) are serviced from the write buffer. If the target data is in the buffer ("write hit"), the old version becomes invalid and the new version is written into the buffer. Otherwise ("write miss"), to admit the data into the buffer, the LRU-selected data is evicted to HDD.

Here, the three types of writes collectively consume the write budget allocated to the workload. In addition to the two explicit write types (① and ③), the flash device inherently experiences ② writes coming from the GC execution. However, in our proposed framework, once the allocated write budget runs out, the three write sources are removed. Primarily, (i) further host writes are redirected to and serviced by the HDD. Also, (ii) further read misses are serviced by the HDD without corresponding data admission into the flash cache, which would prevent read miss-induced writes into the flash cache. The two actions above (i)+(ii) can prevent GC writes from further being generated.

We make a few key assumptions to clearly define our lifetime allocation problem. The problem of fairly allocating three resources – capacity, bandwidth, and lifetime in our case – in a coordinated manner may seem like multi-resource allocation formulations such as dominant resource fairness (DRF) [17]. In fact, our problem is more complex, because of the intricate relationship between allocated capacity (both user and OP) and corresponding bandwidth and write (① host writes, ② GC writes, and ③ read miss-induced writes) intensity; DRF allows for no such relationships. Leaving such generalization as a possible future direction, we take a simpler approach.

Regarding capacity, we first assume that our read cache allocation is based on trying to equalize read hit ratios across

workloads. This can be realized by estimating the working set size of each workload based on a high-precision reuse distance model, which can be done at runtime, as in [2, 3]. We also assume that the remaining user capacity is evenly divided among workloads for their write buffers; while the write buffer size can affect the write miss ratio (and the rate of data eviction to HDD), this does not cause flash writes. For bandwidth, we assume that it is plentiful and is never a bottleneck resource, which may be acceptable in modern flash devices exhibiting up to several GB/s bandwidth [52].

These assumptions above leave us with a fair allocation problem for a single resource – lifetime, i.e., write intensity. Note that writes include host, GC, and read miss-induced writes in our work.

2.4 Other Related Work

Flash device lifetime has been regarded as a first-class resource that needs to be carefully conserved due to its non-renewable nature. One straightforward way to improve flash device lifetime is to reduce the size of original data to be written to the device. Kim et al. [29] employed data deduplication in an SSD, explored possible designs, and investigated deduplication efficiency in terms of lifetime saving and performance overhead. While the data deduplication which works in flash page granularity, Wu et al. [63] attempted to further reduce the number of write operations using fine granular units. In contrast, Lee et al. [33] employed data compression in an SSD, designed the FTL to support the data compression, and analyzed the impact of compression ratios to the performance and lifetime improvement. Li et al. [35] observed that simply compressing data generally results in unused space within page and exploited this to further improve the device lifetime. Zhang et al. [69] also investigated the feasible ways of better utilizing the unused page space.

On the other hand, there have been efforts to make flash devices endure more than advertised/guaranteed. Yadgar et al. [64] employed write-once memory (WOM) code in an SSD and tried to allow multiple writes before erasing the cells. Jeong et al. [24] observed that cell damage depends on the erase voltage level (or erase speed) and proposed erase voltage scaling to reduce cell damage and hence serve more write operations. Jimenez et al. [26] suggested to reduce the bit density of memory cells from multi-level cell (MLC) to single-level cell (SLC), once the MLC device comes to the end of its advertised lifespan.

While all the works above have been investigated under single workload scenarios, they can be employed in consolidated scenarios. Also, their common goal (that is to extend device lifetime) is orthogonal to ours (that is to divide given lifetime to consolidated workloads in a fair manner); so, they can be combined with our proposal.

The lifetime of consolidated flash has been studied in two different settings: soft-partitioned SSD and hard-partitioned SSD. Choi et al. [11] presented a soft-partitioned SSD where a workload's data can be placed into any physical block, but a block is not shared by different workloads. Kim et al. [28] agreed with this design, and further add a function of partitioning OP space to consolidated workloads, as doing so can provide a guaranteed service level to each workload. In contrast, Huang et al. [20] proposed a hard-partitioned SSD that reserves entire flash chip(s) or memory bus(es) for each workload. While this design is very good at providing performance isolation, it needs to regularly swap workloads, since parts of the SSD (assigned to write-intensive workloads) can be worn out more than the others (assigned to workloads with fewer write intensity), which incurs lots of extra writes.

While our ideas of fairly distributing given lifetime among consolidated workloads can be implemented in both settings, we target a soft-partitioned SSD where we can divide OP space and investigate its impact on write allocation decision. In contrast, OP partitioning is not available in hard-partitioned SSDs, as OP is already included in the exclusively-partitioned space. As a result, generally the efficacy of our solution will be smaller in hard-partitioned SSDs.

2.5 Useful Concepts for Allocation and Attribution

We describe two notions from classical economics that we draw upon to develop our solutions.

2.5.1 Max-Min Fairness: One of the most popular, single resource fair allocation strategies is based on max-min fairness [5], which maximizes the minimum allocation received by a participant. The key property offered that is considered fair is the following: an attempt to increase the allocation of a participant is necessarily accompanied by a decrease in the allocation of another participant with an equal or smaller allocation. The max-min fairness allocation can be obtained by the progressive filling [5], which works as follows. It increases all users' allocations at the same rate (from their current allocations, initially zero for all) until some user's demands are fully met or the resource is exhausted. In the former case, after eliminating that user from further consideration, it repeats this process for the remaining users with the remaining resource. The algorithm terminates either when all users' needs are met or the resource has been fully allocated.

2.5.2 Shapley Value: In game theory, Shapley value [54] is a tool for distributing the total gain/value generated by a group of players participating in a coalitional game. Specifically, in a coalitional game where one needs to determine the payoff for each player based on his/her contribution to the total gain, Shapley value provides a *fair way* of doing so by evaluating a range of cases where different subsets of players participate. Out of a coalition set N, the payoff for player i is calculated as follows:

$$\phi_i(V) = \sum_{S \subseteq N \setminus \{i\}} \frac{|S|!(|N|-1)!}{|N|!}(V(S \cup \{i\}) - V(S)), \quad (1)$$

where $V(S)$ is the total expected gain a coalition S can make; here, S is a subset of the original coalition N *excluding* player i. Note that, $V(S \cup \{i\}) - V(S)$ indicates the contribution of

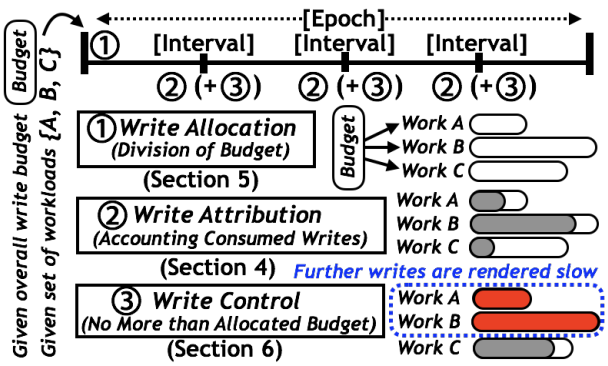


Figure 3. High-level view of our framework. In each and every epoch, (i) the given overall write budget is newly allocated to the given consolidated workloads. At every interval in an epoch, (ii) the total writes consumed by that moment are attributed to the workloads; (iii) if the number of writes attributed to a workload is larger than that allocated to the workload, its further writes are controlled.

player i, when he/she cooperates with a coalition S; and, the contributions of all possible different coalitions are averaged to determine the payoff of player i in N.

Shapley value is considered desirable, as its attribution has the following properties that may be deemed "fair": efficiency, symmetry, null player, and additivity [54]. Because of these properties, Shapley value has been used in other computing contexts such as shared energy usage in datacenters [14, 22, 25] or shared bandwidth in networks [9, 38, 57].

3 Budget-Aware Lifetime Management

3.1 High-Level View

Figure 3 illustrates how our lifetime management framework works in an *epoch*. An epoch is a window in future (of relative workload stationarity) over which fair write allocation decision are made based on predicted workloads. ① At the start of each epoch, the given total write budget is distributed to competing workloads (write allocation). The epoch and the budget for the epoch are given parameters; they can vary, depending on the projected device lifetime (administrator's decision) and workload consolidation scenarios. Workload churn can cause the set of consolidated workloads to change dynamically; some workloads may leave/finish or some may join/start. Whenever there is such a change, our framework ends the current epoch and begins a new epoch.

Each workload is allowed to consume only the allocated writes during the epoch. To this end, our framework introduces a smaller time scale, which is called *interval* – a window in the past over which attribution of writes is done. ② At every interval within the epoch, the overall writes collectively consumed by all workloads at that moment are attributed to each workload (write attribution). ③ If the number of attributed writes is larger than that of the allocated writes for a workload, the workload's future writes are controlled (write control) – further writes are not allowed to get serviced. For the interval, in general, setting it as small as possible (and hence putting as many intervals as possible into an epoch) would be better, because frequent invocations of write attributing (and controlling) can prevent each workload from grossly exceeding its allocated budget. But, the computational overhead of write attribution would pose a limit on the length of an interval.

3.2 Detailed Activities

We describe the three key activities in details:

① Write Allocation (+ Corresponding OP Allocation): We first describe what constitutes a "feasible" allocation; a fair solution would then be chosen from among all feasible solutions based on our chosen "fairness criterion". A feasible allocation must satisfy two constraints. First, the write budget allocations for individual workloads must collectively not exceed the total write budget for the current epoch. Second, the corresponding allocations of OP capacities must collectively not exceed the overall OP capacity. What makes this particularly complex is that the GC writes arising from a workload depend intimately on the OP capacity allocated to it; so, the two constraints must be considered *jointly*. Our write allocation problem can be expressed as follows:

$$\begin{aligned} &Find \quad S_{fair}\{B(1), ..., B(i), ..., B(N_{con})\} \\ &\quad and \quad OP_{fair}\{OP(1), ..., OP(i), ..., OP(N_{con})\} \\ &such\ that \quad B = \sum_{i=1}^{N_{con}} B(i) \quad and \quad OP = \sum_{i=1}^{N_{con}} OP(i), \end{aligned} \tag{2}$$

where B and $B(i)$ are the total write budget and the budget allocated to workload i, respectively; OP and $OP(i)$ are the total OP capacity and the OP capacity allocated to workload i, respectively. N_{con} is the number of consolidated workloads. Among feasible allocations (satisfying the two constraints above), our goal is to find S_{fair} and the corresponding OP_{fair}, which will be discussed in Section 5.

One challenge in exploring the above problem is that we need to be aware of the number of GC writes each workload is likely to generate under any given OP capacity. To obtain (estimate) the number of GC writes, we employ an existing analytic model to *estimate* its WAF value. As the number of host writes for each workload is easy to acquire, once one obtains the WAF value from the model, the number of GC writes can be obtained. In our current implementation, we employ an accurate WAF value estimation model [13] (for more details, refer to Equation 20 in [13]). Our employed model approximates Greedy performance with the following important parameters: (i) the allocated OP capacity, (ii) the number of pages in a block, and (iii) the fraction (f) of hot data and the I/O rate (r) on such hot data. For (iii), we first build "page data vs the number of updates" for each I/O trace of Table 4; we then manually find a set of possible (f, r) pairs that best characterize the hot-cold data separation, and finally pick the one whose performance is the most comparable to that of the actual simulation.

$N \setminus \{W1\} = S = \emptyset, \{W2\}, \{W3\}, \{R1\}, \{W2, W3\}, \{W2, R1\}, \{W3, R1\}, \{W2, W3, R1\}$; ϕ_{W1} = ❶ + ❷ + ❸ + ❹ + ❺ + ❻ + ❼ + ❽	
❶ $\frac{1}{4} \times \{V(W1) - V(\emptyset)\}$	❷ $\frac{1}{12} \times \{V(W1, W2) - V(W2)\}$
❸ $\frac{1}{12} \times \{V(W1, W3) - V(W3)\}$	❹ $\frac{1}{12} \times \{V(W1, R1) - V(R1)\}$
❺ $\frac{1}{12} \times \{V(W1, W2, W3) - V(W2, W3)\}$	❻ $\frac{1}{12} \times \{V(W1, W2, R1) - V(W2, R1)\}$
❼ $\frac{1}{12} \times \{V(W1, W3, R1) - V(W3, R1)\}$	❽ $\frac{1}{4} \times \{V(W1, W2, W3, R1) - V(W2, W3, R1)\}$

Table 2. Shapley value accounts the number of writes to $W1$ (ϕ_{W1}) by exploring all possible sub-coalitional games and considering the marginal contribution of $W1$ to the games. The same approach can be used for all other workloads.

② **Write Attribution:** To prevent each workload from consuming more writes than its allocated budget in an epoch, there is a need for accounting the total number of writes and identifying how many writes each workload consumes in the middle of the epoch. Among various possible ways of accounting the generated writes, we are interested in the one that provides a notion of *fairness*. So, motivated by its fairness properties, we propose to employ Shapley value in our write attribution process; and, we also compare our Shapley value-inspired accounting strategy with the ones based on two other likely-used accounting policies, which will be discussed in Section 4.

③ **Write Control:** Depending on the attribution results, if necessary, the host writes of workloads whose write budget run out are prevented until the end of the epoch. Note that this is possible in the caching-tier of storage hierarchy, while the main-storage should process all writes coming from the host/workloads. Specifically, we propose to make the further writes beyond the allocated budget *bypass* our target device of the caching-layer and get serviced from the next-layer HDD storage. The service time of these writes would be a bit slowed down, which is referred to as "writes rendered slow" or "slowed writes". Our write control mechanism will be discussed in detail in Section 6.

4 Our Proposed Write Attribution Strategy

4.1 Applying Shapley Value to Write Attribution

Our write *accounting* problem is to attribute the total number of serviced writes from a flash device (both host and GC writes) to the consolidated workloads. We analogize "write generation" in our consolidated device setting to "benefit creation" in a coalitional game. Specifically, we analogize consolidating multiple workloads to a cooperative game wherein we treat the total number of writes generated from the consolidation as corresponding to the total surplus in the game. As Shapley value can determine the payoff of each player, it can help one attribute writes to different workloads.

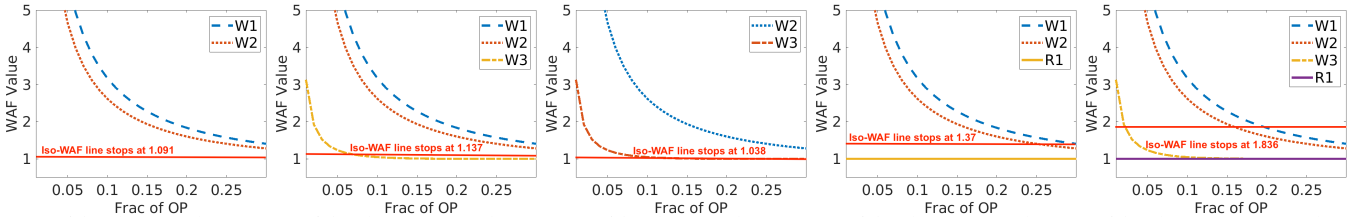
To demonstrate Shapley value in a more practical setting, we assume a flash device where a set of four workloads are consolidated (4C1 of Table 5; details of individual workloads – W1, W2, W3, and R1 – are shown in Table 4). Here, we can define our game as follows: $N = \{W1, W2, W3, R1\}$ and $V(S)$ is the total number of writes when workloads in S (a subset of N) are consolidated. Table 2 describes how Shapley value derives the number of accounted writes to one of our workloads, $W1$. There are 8 possible subsets S that exclude $W1$ (i.e., $N \setminus \{W1\}$), and for each subset, the marginal contribution of $W1$ to the increase in the write count is determined by comparing the subset with $W1$ with the subset without $W1$ (❶~❽). Then, the marginal contributions across all possible subsets are *averaged*, based on the cardinalities and combinations of possible subsets ($\sum$ ⓝ). For all other workloads, their accounted writes can be calculated in a similar fashion.

4.2 Challenge: Unobserved Subsets

A potential difficulty in employing Shapley value in our problem is the *unobserved* subsets, of which total writes are unknown online. Actually, we know the total number of writes only for the case where *all* the workloads are consolidated – $V(W1, W2, W3, R1)$; however, we do not know the number of writes in any other subset.

To identify the total number of writes in an unobserved subset, we need to know the OP partition of the subset; once we know the OP partition, we can obtain the total number of writes by using the number of host writes and the WAF model (introduced in Section 3.2) of each workload. Let us consider, as an example, the calculation of ❷ in Table 2, where we need to get two unobserved sets, $V(W1, W2)$ and $V(W2)$. To obtain the latter where $W2$ is executed alone, there is no issue of partitioning capacity as we reserve all the device capacity (except for its read cache and write buffer) as its OP. However, for the former where $W1$ and $W2$ are consolidated together, we need to decide how to partition the OP capacity between them.

Our Heuristic: To this end, we propose an OP allocation strategy, called *Iso-WAF*, which is based on our belief that a fair allocation would *equalize* write amplifications across all consolidated workloads. Specifically, this strategy finds an OP partitioning where WAF values of all workloads are equal, by keeping increasing the Iso-WAF line starting from one, and stops at the smallest WAF value that satisfies the OP capacity constraint. Figure 4 provides an example application of Iso-WAF to five different subsets, where Iso-WAF line stops at different WAF values or capacity partitions. Figures 4b and 4c show the two cases (❺ of Table 2) which the Shapley value write accounting explores. One can see that the Iso-WAF value of $\{W1, W2, W3\}$ (1.137) is larger than that of $\{W2, W3\}$ (1.038). This is because (i) the available OP capacity of the former decreases due to the consolidation of $W1$ (i.e., due to the cache/buffer size of $W1$) and (ii) the OP capacity should be shared by three workloads, which collectively make the Iso-WAF line go up. We also want to compare $\{W1, W2, R1\}$ with $\{W1, W2\}$ through Figures 4d and 4a (which can be used in the process of calculating ϕ_{R1}).



(a) $V(W1, W2)$ **(b)** $V(W1, W2, W3)$ **(c)** $V(W2, W3)$ **(d)** $V(W1, W2, R1)$ **(e)** $V(W1, W2, W3, R1)$

Figure 4. Our Iso-WAF approach finds an OP partition which equalizes WAF values across all consolidated workloads. In each of OP vs WAF graphs, the Iso-WAF line (increasing from one) stops at the smallest WAF value that satisfies the OP constraint.

Total Writes	17,337,348		
Accounting Policy	**Generated**: directly generated by each **Even**: evenly accounted over non-read-only **Shapley**: Shapley value over all		
Workloads	**Generated**	**Even**	**Shapley**
W1	5,970,700	5,779,116	5,362,803
W2	10,389,068	5,779,116	8,255,882
W3	977,580	5,779,116	1,770,269
R1	0	0	1,948,394

Table 3. Different policies lead to different accounting results. Our approach accounts writes to $R1$ as well.

The significant difference in the Iso-WAF values between these two subsets originates from the large read cache size of $R1$. Note that, even though $R1$ does *not* need any OP space, our Iso-WAF partitioning still considers its read cache size, which causes a decrease in the available OP capacity of the system. That is, mere existence of a read-only workload reduces the total OP capacity, which in turn affects the write behavior of our write-intensive workloads. Figure 4e depicts the case where all four workloads are consolidated; with a further decreased OP availability, the Iso-WAF value increases significantly.

4.3 Accounting Results and Observations

After executing the set of workloads (4C1 of Table 5) on a device (details of the experimental setup can be found in Section 7.1), we use our Shapley value-inspired accounting for attributing the total writes to the individual workloads. Table 3 compares three different polices and shows how each policy accounts the total number of writes (17,337,348) to our four workloads. The first policy (*Generated*) monitors all activities (host writes and GC writes) of each workload and counts the exact number of writes it generates; the philosophy behind this policy is that a workload should be responsible for exactly what it generates. The second policy (*Even*) accounts the same number of writes to all non-read only workloads; this assigns an even responsibility to all the non-read only workloads. The last policy (*Shapley*) employs Shapley value and calculates ϕ of each workload.

Our Finding: One interesting observation is that Shapley accounts a significant number of writes to $R1$ (whereas Generated and Even attribute no writes to $R1$). As stated earlier, this is because our Shapley value-inspired accounting holds $R1$ responsible for the writes generated by its consolidated (co-runner) workloads. Specifically, the read cache of $R1$ takes a large amount of storage capacity, which reduces the OP capacities assigned to the consolidated workloads, and this in turn increases the number of amplified writes from them. As an example, let us investigate $W2$: it generates over 65% of the total writes (see the number of writes accounted to $W2$ by Generated), as (i) it issues a lot of host writes and (ii) it also significantly increases the number of amplified writes. As $W2$ is sensitive to its assigned OP capacity (Figure 4e), if it had more OP capacity, the number of its amplified writes and the total number of writes would decrease. In our consolidated setting however, $R1$ prevents $W2$ from grabbing a large OP capacity and reducing the number of its amplified writes, which contributes to the total number of writes. Interestingly, the number of amplified writes accounted to $R1$ is larger than that accounted to $W3$. This is because the impact of high capacity taken by $R1$ is more significant than the direct contribution of $W3$ to the total number of writes. Note also that $W3$ takes a small capacity for both cache and OP, which means it makes a marginal contribution to the total number of writes.

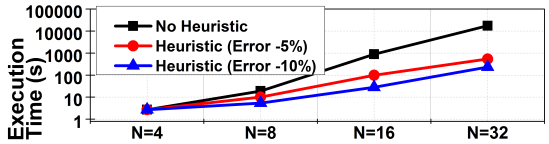


Figure 5. Execution time of our Shapley value-inspired attribution under varying number (N) of consolidated workloads.

4.4 Scalability: When N is Large

Since in our example N is 4 (i.e., the number of consolidated workloads is 4), the number of possible subsets our Shapley value-inspired accounting should consider is only $2^4 = 16$, which does not need much computation time. However, as the degree of consolidation increases, the number of subsets exponentially increases. For example, in an aggressively-consolidated setup (where N is 40), the number of subsets our accounting needs to consider is 2^{40}, which can bring significant computation overheads.

Our Heuristic: To make our accounting technique feasible and scalable in high consolidation scenarios, we propose a heuristic that considers only a fraction of all possible sets in calculating the Shapley value. The motivation behind our heuristic is that there is no need to consider the subsets whose sizes are small. This is because a small subset generates no or few amplified writes, as each participating workload can have a large amount of OP capacity, and this makes its WAF value close to 1. Let us consider, as an example, a 1TB SSD where N workloads are consolidated, and

a subset $S = \{W1, W2, W3, R1\}$ of N. For this subset, each workload could have hundreds GB for its OP and the WAF value of each converges to 1. Consequently, the total number of writes in this subset ($V(S)$) is just the sum of the host writes of the participating workloads, which can be obtained without any calculation. Among all possible subsets, the subsets whose sizes are equal to or smaller than 25 fall into this category, which can be skipped in the Shapley value calculation. Note that the total number of such sets is $\sum_{i=1}^{25} \binom{40}{i}$, which corresponds to 96% of the total number of possible sets. Consequently, this heuristic requires us explore in detail only 4% of the entire subsets. However, this can be still regarded as significant, depending on the available compute resources. If a slight error in the accounting result is acceptable, we can further reduce the computation time by pruning more subsets; the error is defined as the difference in the attributed writes between considering all possible sets and subsets. For example, if we drop the subsets whose sizes are equal to or smaller than 30 from consideration, the number of subsets our accounting technique needs to explore would be $\sum_{i=31}^{40} \binom{40}{i}$, which corresponds to only 0.03% of the total number of possible subsets! According to our analysis, if we drop more subsets, e.g., subsets whose sizes are equal to or smaller than 34 and 35, the number of remaining subsets that our accounting technique explores would be only 760,099 and 102,091, respectively. Such an aggressive subset pruning brings 4% and 7% error in accounting, compared to the exact Shapley value calculation that explores *all* subsets.

Computation Overhead Analysis: We executed our Shapley value-inspired attribution (implemented using MATLAB program) on 3.4 GHz Intel Core i7-based host system. Figure 5 plots the CPU computation times of three different strategies: the base Shapley value (No-Heuristic), our heuristic allowing the error up to 5% (Error-5%), and our heuristic allowing the error up to 10% (Error-10%). The computation time of No-Heuristic increases exponentially as the number of consolidated workloads increases (note that the Y-axis is in log-scale). When N=16 and 32, the computation times (over 15 minutes and 1 hour) are not acceptable, when the sizes of epoch (1 hour) and interval (15 minutes) are considered. In contrast, Error-5% and Error-10% significantly reduce the computation time; when N=32, it takes only 4-8 minutes. This indicates that our heuristic makes Shapley value-inspired attribution a *scalable* option.

5 Write Allocation Strategies

5.1 Baseline Strategies

5.1.1 EVEN – Partitioning OP Evenly: When considering the notion of fairness, the first choice that comes to mind is to (i) evenly partition the OP capacity among workloads and (ii) estimate the number of writes for each workload with this even OP share. Specifically, each non-read-only workload is given an "equal share" of the available OP capacity, which is expressed as follows:

$$OP(i) = OP/(N_{con} - N_{read}), \quad (3)$$

where OP and $OP(i)$ are the total OP capacity and the OP allocated to non-read-only workload i, respectively, and N_{con} and N_{read} are the numbers of the consolidated workloads and read-only workloads, respectively.

Based on this even OP partitioning, one can estimate the total number of writes ($\hat{W}$) each workload generates, and this could be set as the write budget allocated to each workload ($B(i)$ for workload i), which can be calculated as follows:

$$B(i) = \hat{W}(i) = H(i) \times A(i) = H(i) \times f_i(OP(i)), \quad (4)$$

where $H(i)$ and $A(i)$ are the number of host writes and the WAF value of workload i, respectively. Also, f_i and $OP(i)$ denote, respectively, the WAF model function and the allocated OP capacity of workload i.

Unfortunately, the even OP partitioning is *not* always feasible. The problem is that the sum of estimated number of writes can be larger than the total write budget, which can be expressed as follows:

$$\sum_{i=1}^{N_{con}} \hat{W}(i) = \sum_{i=1}^{N_{con}} B(i) > B, \quad (5)$$

Note that this scenario can frequently occur, since even OP partitioning *lifetime-agnostic*.

5.1.2 MMF – Partitioning OP to Achieve Max-Min Fair Division of Total Budget: To address the above shortcoming of even partitioning, we employ the notion of max-min fairness to make the sum of estimated writes fit within the total write budget. Specifically, as EVEN strategy does, this strategy (i) begins with the even OP partitioning (Equation 3) and (ii) the estimation of the total number of writes each workload would generate under the even OP share (Equation 4). However, then, it (iii) uses the progressive filling algorithm [5] to distribute the total budget to consolidated workloads based on their estimated number of writes, which results in the max-min fairness division of the total budget. As a result, the budget allocated to workload i, ($B(i)$), can be calculated as follows:

$$B(i) = \max \min(\hat{W}(i), B), \quad (6)$$

where $\hat{W}(i)$ and B are the write budget allocated to workload i and the total write budget, respectively.

Based on the max-min fairness-based write allocation, finally, this strategy (iv) re-partitions the total OP capacity. The OP capacity assigned to workload i, $OP(i)$, is obtained by back-calculating the allocated budget ($B(i)$) as a function of the OP capacity, as expressed below:

$$OP(i) \ \ s.t. \ \ B(i) = H(i) \times f_i(OP(i)), \quad (7)$$

where $H(i)$ and f_i are the number of host writes and the WAF model function of workload i, respectively. Note that, if the sum of the estimated writes in all workloads is less than or equal to the total write budget (i.e., the opposite case of Equation 5), this MMF strategy would reduce to EVEN. That is, the max-min fairness-based budget partitioning and OP partitioning would be the same as those of the EVEN

strategy (Equations 4 and 3). To sum up, through (i) to (iv), the MMF strategy makes up for the shortcomings of the EVEN strategy.

5.1.3 Iso-WAF – Partitioning OP to Equalize Write Amplification Factor Values: This strategy is rooted in the ideas that a fair allocation would *equalize write amplifications* across all consolidated workloads. So, this strategy (i) finds an OP partition where the WAF values of all workloads (excluding the read-only ones for which WAF is zero) are equal. Among numerous possible partitions, it then selects the one where the sum of assigned OP capacities is equal to the total OP capacity. The OP capacity assigned to workload i, $OP(i)$, is determined by the following constraints:

$$OP(i) \;\; s.t. \;\; A(i) = A(j) \;\; and \;\; \sum_{k=1}^{N_{con}-N_{read}} OP(k) = OP, \quad (8)$$

where $A(i)$ is the WAF value of workload i and OP is the total OP capacity. N_{con} and N_{read} are the number of consolidated workloads and read-only workloads in them, respectively. Finding the OP partition is easy – based on the OP-WAF graph, we keep increasing the WAF line starting from 1, and stop at the *smallest WAF value* that satisfies $\sum_{k=1}^{N_{con}-N_{read}} OP(k) = OP$. Note that, as the Iso-WAF line goes up, the OP capacity that each workload can have decreases, and at some point, the sum of the OP capacities meets the total OP capacity.

After that, this strategy (ii) finds a budget allocation where the WAF values of all workloads (excluding read-only ones) are equalized. For this, it selects a write allocation where the sum of the estimated writes is equal to the total write budget. The write budget allocated to workload i, $B(i)$, is determined by the following constraints:

$$B(i) \quad s.t. \;\; A(i) = A(j) \;\; and \;\; \sum_{k=1}^{N_{con}-N_{read}} \hat{W}(k) = B, \quad (9)$$

where $\hat{W}(k)$ and B are the estimated number of writes of workload k and the total write budget, respectively. Finding the budget allocation is carried out as follows. Based on the OP-WAF graph, we keep decreasing the WAF line starting from a large number (e.g., 5), and stop at the *smallest WAF value* that satisfies $\sum_{k=1}^{N_{con}-N_{read}} \hat{W}(k) = B$. As the Iso-WAF line goes down, the number of estimated writes each workload generates decreases, and at some point, the sum of estimated writes meets the total write budget.

5.2 Proposed Strategy: SV – Using Shapley Value

Motivated by the notion of fairness our Shapley value-inspired accounting provides, we propose to employ the same mechanism in the budget allocation as well. Note that the write accounting deals with attributing the "already-consumed writes" to participating workloads, whereas the budget allocation deals with assigning the "expected number of writes" a workload may generate to the workload *in advance*. Specifically, the strategy uses Iso-WAF method (Section 5.1.3) for both (i) OP partitioning and (ii) write allocation. Then, assuming that the allocated write budget of a workload is the total number of writes it generates, the strategy (iii) applies Shapley value to redistribute the total budget. Note that this SV strategy allocates write budget to read-only workloads, as the Shapley value-inspired accounting does.

Label	Workload	RD/WR Ratio	RD Cache(GB)	# Host WR
W1	exchange	32.7/67.3	7.53	3,252,015
W2	msnfs	64.1/35.9	21.41	5,658,534
W3	moodle	93.5/06.5	4.93	532,450
W4	hm	32.5/67.5	17.39	5,989,990
W5	prn	19.8/80.2	11.52	14,040,432
R1	readonly1	100.0/00.0	15.00	0
R2	readonly2	100.0/00.0	7.50	0

Table 4. Characteristics of individual workloads.

Degree Consol.	Consol. Scenario	Individual Workload W1	W2	W3	W4	W5	R1	R2
2	2C1	●	●					
	2C2			●	●			
	2C3		●			●		
	2C4	●					●	
	2C5		●		●			
3	3C1	●	●	●				
	3C2			●	●	●		
	3C3		●		●		●	
	3C4			●	●			●
	3C5		●	●				●
4	4C1	●	●	●			●	
	4C2	●	●		●	●		
	4C3		●			●	●	●
	4C4	●		●	●			●
	4C5		●	●	●		●	
	4C6		●	●	●	●		

Table 5. Tested consolidation scenarios, each of which combines individual workloads of Table 4.

6 Our Write Control Strategy

Different from the prior SSD throttling studies [32, 36], the write-control mechanism in our framework works as follows:
• Once the framework detects that a workload consumes all of the allocated budget to it, all the host writes the workload issues *bypass* the flash cache and are served directly by the HDD store. As a result, no more writes (including GC writes) from that workload are served by the flash device, while it continues to be executed using the HDD with increased write response times. Note however that, all read hits continue to be served from the read cache; read misses are served from the HDD without any cache admission (miss-induced write).
• Read-only workloads issue no host writes; so, even if their attributed writes cannot meet their allocated budgets, the control mechanism cannot take any action.

7 Evaluation

7.1 Experimental Methodology

Framework: We implemented MMF, Iso-WAF and SV mechanisms using MATLAB; and we also designed a consolidated SSD working with our framework, using the DiskSim [7] simulator with the SSD extensions [1]. Specifically, we added mechanisms for supporting workload consolidation (Section 2.2) to the simulator – workload-aware, block-granularity soft-partitioned, and per-workload GC mechanisms. The budget allocations obtained from the MATLAB program are given to the simulator, and the simulation results are used for write accounting in the MATLAB program.

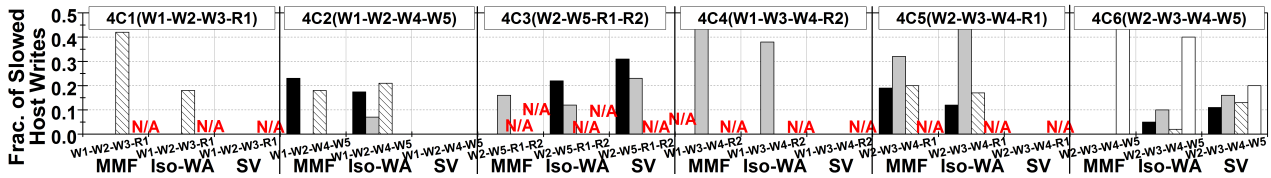


Figure 6. Fraction of slowed writes to total host writes (under 3 strategies in 6 scenarios). R1/R2 are not applicable (N/A).

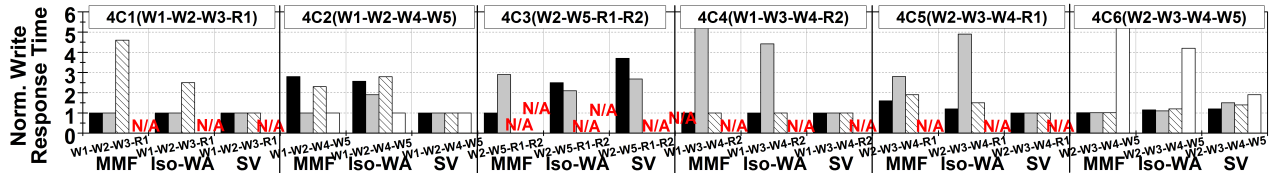


Figure 7. Write response times normalized to the case where all writes are handled by SSD (under 3 strategies in 6 scenarios).

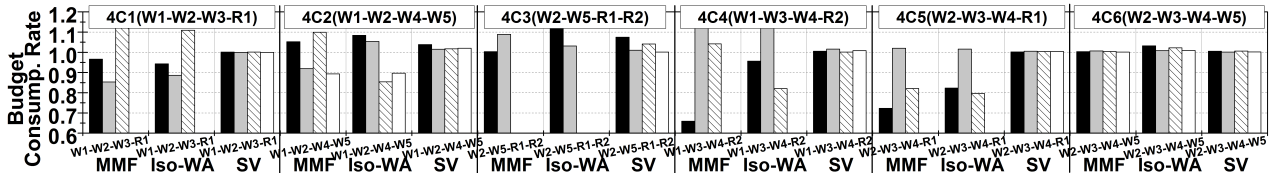


Figure 8. Fraction of consumed writes to allocated budget (under 3 strategies in 6 scenarios).

System Configuration: We assume a 80GB SSD, which consists of 8 packages, each having 10,240 blocks, and a block consists of 256 4KB pages. We also assume that the total OP capacity is 5% of total storage capacity (i.e., 4GB). For the total user capacity (i.e., 76GB), the read cache and write buffer sizes vary depending on workload consolidation scenarios. Specifically, the read cache for each individual workload is allocated such that its read hit ratio is 95%; and, all the remaining user capacity is equally partitioned for their write buffers. For GC and WL, we employ greedy [8] and static [41] algorithms, respectively. For the HDD (paired with the SSD), we employ a latency model from [30] – 3.6ms HDD read/write latency. For the SSD latencies [40], we set 200us, 2.6ms, and 3ms for read, write, and erase, respectively

Workloads: We used 5 real-world (write-intensive) workloads from [43] and [4] and 2 synthetic (read-only) workloads, which are listed in Table 4. Considering the capability of our target SSD, we constructed various 2 to 4-workload consolidation scenarios by mixing the individual workloads, which are listed in Table 5. Note that, consolidating more workloads is overkill, since doing so makes the collective write intensities and user capacity demands of consolidated workloads beyond the SSD capability. We mainly report the results of 4-workload consolidation scenarios, each of which collectively needs a lot of writes; so, a fair write allocation is more important. Note however that, the same analysis can be applied to 2 and 3-workload scenarios.

The epoch size is set to *1 hour* (note that this can be tuned by the administrator); so, we extracted multiple 1-hour traces from each individual workload and combined them for consolidation workloads. Considering our epoch size (1 hour) and computational overhead of write attribution, the interval size is set to *15 minutes*. The total budget for each epoch is tightly set to *5M page writes* by considering the average writes our workload consolidation scenarios generate.

Systems Evaluated: We compared the following three write allocation strategies (note that EVEN is not feasible under the small budget, as discussed in Section 5.1.1):

- **MMF (Section 5.1.2):** This employs the max-min fairness to divide the total budget, assuming an even OP partitioning.
- **Iso-WAF (Section 5.1.3):** This is based on our proposed Iso-WAF mechanism for write allocation and OP partitioning.
- **SV (Section 5.2):** This applies Shapley value to the budget divided by the Iso-WAF strategy.

Metrics: To evaluate the *lifetime fairness* of these three strategies, we focused on the following three metrics:

- **Fraction of Slowed Writes (Section 7.2):** This metric indicates the ratio of the slowed writes to the total host writes. If a very small budget is allocated to a workload, it would experience many slowed writes; so, the smaller this metric *across workloads*, the fairer the write allocation. Note that this metric is not applicable to read-only workloads.
- **Write Response Time (Section 7.2):** This metric captures the impact of the slowed writes on the workload performance. Since all controlled writes get serviced from the HDD whose latency is longer than that of SSD, the shorter the write response times *across workloads*, the fairer the allocation. The values are *normalized* to the case where all host writes get serviced from the SSD. For read performance, we assume that read cache sizes are determined such that read hit ratios are equalized across workloads (see Section 2.3).
- **Budget Consumption Rate (Section 7.3):** This metric represents the ratio of consumed writes to allocated writes. If a budget allocated to a workload is more than actually needed, many writes would remain unused, which leads to a small value of this metric. Hence, the smaller this metric *across workloads*, the worse the write allocation.

Individual workloads of 4C2	W1	W2	W4	W5
❶ Using our WAF model	20,013	51,125	5,610	11,369
❷ Using actual GC information	23,875	45,772	5,233	12,450
Error (❶–❷) ÷❷ × 100	-16.17%	11.69%	7.21%	-8.68%

Table 6. Our model-based vs actual GC value-based write attribution results in a sample interval of 4C2 scenario.

7.2 Fraction of Slowed Writes and Response Time

Figure 6 compares the three systems in terms of the fraction of slowed writes to the total host writes. Bars in the graph are omitted for read-only workloads (R1 and R2), or if the count of slowed writes is zero.

• A fair allocation tends to suppress the slowed writes across all workloads. MMF and Iso-WAF make one or more workloads experience a lot of slowed writes by allocating too small budgets to them. In contrast, SV avoids such cases by fairly distributing the total budget based on the needs.

• One might think that SV makes an unfair allocation for 4C3 as the two write-intensive workloads (W2 & W5) are rendered significantly slow, due to their small budgets. However, this is because SV allocates a large fraction of the total budget to the read-only workloads, and this causes a decrease in the write budget allocated to the write-intensive workloads. In contrast, MMF and Iso-WAF allocate zero writes to the two read-only workloads, although lots of writes are attributed to them from the first interval; actually, they are not fair.

• Figure 7 plots the write response times of the individual workloads under different allocation strategies. All values are *normalized* to the case where all host writes get serviced from the SSD. The more slowed writes, the longer write response times are experienced; and, the shorter the write response times across workloads, the fairer the allocation.

7.3 Budget Consumption Rate

The metrics above (Section 7.2) cannot capture the case where a very large budget is allocated to one or more workloads. The budget consumption rate metric on the other hand may detect such scenarios, and is plotted in Figure 8.

• MMF and Iso-WAF make many individual workloads have a value smaller than 1 by allocating them more writes than they need. In contrast, SV redistributes these surplus writes over the read-only workloads.

• It can be observed that R1 and R2 under MMF and Iso-WAF have a value of 0; this is because these two strategies do not allocate any write to read-only workloads, while writes are attributed to them. In contrast, SV allocates writes to read-only workloads as well; consequently, in all consolidation scenarios, all individual workloads have a value of around 1.

• Many individual workloads have values that are larger than 1 – they consume more than their allocated budgets. This is because the write accounting and controlling are periodically performed (every interval); so the number of consumed writes may already exceed the number of allocated writes. Under MMF and Iso-WAF, workloads may have high values of 1.1-1.2, because few writes are allocated to them, and in turn, they can easily consume more writes beyond allocated budget before the attribution detects it. However, our SV has values of at most 1.01; SV allocates budget appropriately and attribution/controlling can be done on time.

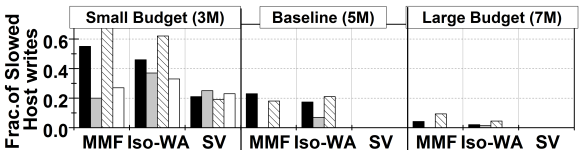


Figure 9. Fraction of slowed writes under the three different strategies in 4C2 scenario under varying total budget sizes.

7.4 Budget Size Sensitivity

To evaluate how the three different strategies work under different budget sizes, we observed the fractions of slowed writes to total host writes using 4C2 scenario, under under small, tight (baseline), and large budget sizes (Figure 9). When a large budget is prescribed, MMF and Iso-WAF may be viable options for fair write allocation, as the allocated budget to each workload is still large. However, MMF and Iso-WAF fail to regulate the values under a small budget; in contrast, SV suppresses the values and makes them close to each other. In general, less the budget, better our SV works.

7.5 Model Error Analysis

To evaluate the trustability of our employed WAF model [13], using a sample interval from a consolidation scenario (4C2 of Table 4), we compared the attribution results of our model-based strategy with an ideal system. For the latter, we actually executed all 16 (2^4) subsets in the Shapley value calculation using our simulation framework. Table 6 shows that our model-based write attribution results differ from those of the ideal system, due to the accuracy of the employed model. While some errors introduced, our WAF model-based framework is quite effective in write attribution and allocation.

8 Conclusions

We identified a fundamentally novel problem related to flash lifetime management where multiple workloads are consolidated on a single flash device. Our problem is based on viewing flash lifetime (equivalently total writes) as a *first-class resource* that needs to be carefully and explicitly managed. In particular, we explored what a fair allocation might mean for this resource. We presented three baselines and a novel strategy, which is inspired by the idea of the Shapley value. Using real-world traces, we demonstrated the superiority of our technique over baselines in helping the consolidated workloads share the available lifetime fairly and efficiently.

Acknowledgments

We thank Peter Desnoyers, our shepherd, and the anonymous reviewers for their valuable feedback. This research is supported in part by NSF grants 1822923, 1439021, 1629915, 1626251, 1629129, 1763681, 1526750, 1439057, and 1717571. Dr. Jung's research is in part supported by NRF 2016R1C1B2015312, DOE DEAC02-05CH11231, NRF 2015M3C4A7065645, NRF 2017R1A4A1015498 and KAIST start-up grant.

References

[1] Nitin Agrawal, Vijayan Prabhakaran, Ted Wobber, John D. Davis, Mark Manasse, and Rina Panigrahy. 2008. Design Tradeoffs for SSD Performance. In *USENIX ATC*.

[2] Saba Ahmadian, Onur Mutlu, and Hossein Asadi. 2018. ECI-Cache: A High-Endurance and Cost-Efficient I/O Caching Scheme for Virtualized Platforms. In *SIGMETRICS*.

[3] Dulcardo Arteaga, Jorge Cabrera, Jing Xu, Swaminathan Sundararaman, and Ming Zhao. 2016. CloudCache: On-demand Flash Cache Management for Cloud Computing. In *USENIX FAST*.

[4] Dulcardo Arteaga and Ming Zhao. 2014. Client-side Flash Caching for Cloud Systems. In *SYSTOR*.

[5] Dimitri Bertseka and Robert Gallager. 1992. Data Networks. In *Prentice-Hall Inc.*

[6] Matias Bjorling, Javier Gonzalez, and Philippe Bonnet. 2017. LightNVM: The Linux Open-Channel SSD Subsystem. In *USENIX FAST*.

[7] John S. Bucy, Jiri Schindler, Steven W. Schlosser, and Gregory R. Ganger. 2008. The DiskSim Simulation Environment Version 4.0 Reference Manual. In *CMU-PDL-08-101*.

[8] Werner Bux and Ilias Iliadis. 2010. Performance of Greedy Garbage Collection in Flash-Based Solid-State Drives. In *Journal of Performance Evaluation, VOL. 67, Issue. 11*.

[9] Jianfeng Cai and Udo Pooch. 2004. Allocate Fair Payoff for Cooperation in Wireless Ad Hoc Networks Using Shapley Value. In *IPDPS*.

[10] Alan Chen. 2015. Integrating Cooperative Flash Management with SMR Technology for Optimized Tiering in Hybrid Systems. In *Storage Developer Conference*.

[11] Samsung (Changho Choi). 2017. AutoStream: Automatic Stream Management for Multi-stream SSDs in Big Data Era. In *Storage Developer Conference*.

[12] Wonil Choi, Bhuvan Urgaonkar, Mahmut Kandemir, and Myoungsoo Jung. 2019. Fair Resource Allocation in Consolidated Flash Systems. In *USENIX HotStorage*.

[13] Peter Desnoyers. 2012. Analytic Modeling of SSD Write Performance. In *SYSTOR*.

[14] Mian Dong, Tian Lan, and Lin Zhong. 2014. Rethink Energy Accounting with Cooperative Game Theory. In *MobiCom*.

[15] Assaf Eisenman, Asaf Cidon, Evgenya Pergament, Or Haimovich, Ryan Stutsman, Mohammad Alizadeh, and Sachin Katti. 2019. Flashield: a Hybrid Key-value Cache that Controls Flash Write Amplification. In *NSDI*.

[16] Abhinav Garg. 2011. Cloud Computing for the Financial Services Industry. In *Sapient Global Markets*.

[17] Ali Ghodsi, Matei Zaharia, Benjamin Hindman, Andy Konwinski, Scott Shenker, and Ion Stoica. 2011. Dominant Resource Fairness: Fair Allocation of Multiple Resource Types. In *NSDI*.

[18] Javier Gonzalez. 2018. Denali Open-Channel SSDs. In *Flash Memory Summit*.

[19] Mohammad Hedayati, Kai Shen, Michael L. Scott, and Mike Marty. 2019. Multi-Queue Fair Queuing. In *USENIX ATC*.

[20] Jian Huang, Anirudh Badam, Laura Caulfield, Suman Nath, Sudipta Sengupta, Bikash Sharma, and Moinuddin K. Qureshi. 2017. FlashBlox: Achieving Both Performance Isolation and Uniform Lifetime for Virtualized SSDs. In *USENIX FAST*.

[21] Ronnie Huang. 2016. Open-Channel SSDs and Host-Based FTLs. In *Flash Memory Summit*.

[22] Mohammad A. Islam and Shaolei Ren. 2016. A New Perspective on Energy Accounting in Multi-Tenant Data Centers. In *USENIX CoolDC*.

[23] Mike Jadon. 2015. Replacing the FTL with Cooperative Flash Management. In *Flash Memory Summit*.

[24] Jaeyong Jeong, Sangwook Shane Hahn, Sungjin Lee, and Jihong Kim. 2014. Lifetime Improvement of NAND Flash-based Storage Systems Using Dynamic Program and Erase Scaling. In *USENIX FAST*.

[25] Weixiang Jiang, Fangming Liu, Guoming Tang, Kui Wu, and Hai Jin. 2017. Virtual Machine Power Accounting with Shapley Value. In *ICDCS*.

[26] Xavier Jimenez, David Novo, and Paolo Ienne. 2013. Pheonix: Reviving MLC Blocks as SLC to Extend NAND Flash Devices Lifetime. In *DATE*.

[27] Bryan S. Kim. 2018. Utilitarian Performance Isolation in Shared SSDs. In *USENIX HotStorage*.

[28] Jaeho Kim, Donghee Lee, and Sam H. Noh. 2015. Towards SLO Complying SSDs Through OPS Isolation. In *USENIX FAST*.

[29] Jonghwa Kim, Sangyup Lee, Ikjoon Son, Jongmoo Choi, ChoongHyun Lee, Sungroh Yoon, Hu ung Lee, Sooyong Kang, Youjip Won, and Jaehyuk Cha. 2012. Deduplication in SSDs: Model and Quantitative Analysis. In *MSST*.

[30] Youngjae Kim, Aayush Gupta, Bhuvan Urgaonkar, Piotr Berman, and Anand Sivasubramaniam. 2011. HybridStore: A Cost-Efficient, High-Performance Storage System Combining SSDs and HDDs. In *MASCOTS*.

[31] Ricardo Koller, Ali Jose Mashtizadeh, and Raju Rangaswami. 2015. Centaur: Host-Side SSD Caching for Storage Performance Control. In *ICAC*.

[32] Sungjin Lee, Taejin Kim, Kyungho Kim, and Jihong Kim. 2012. Lifetime Management of Flash-Based SSDs Using Recovery-Aware Dynamic Throttling. In *USENIX FAST*.

[33] Sungjin Lee, Jihoon Park, Kermin Fleming, Arvind, and Jihong Kim. 2011. Improving Performance and Lifetime of Solid-State Drives Using Hardware-Accelerated Compressions. In *IEEE Transactions on Computer Electronics, Vol.57, No.4*.

[34] Cheng Li, Philip Shilane, Fred Douglis, Hyong Shim, Stephen Smaldone, and Grant Wallace. 2014. Nitro: A Capacity-Optimized SSD Cache for Primary Storage. In *USENIX ATC*.

[35] Jiangpeng Li, Kai Zhao, Xuebin Zhang, Jun Ma, Ming Zhao, and Tong Zhang. 2015. How Much Can Data Compressibility Help to Improve NAND Flash Memory Lifetime?. In *USENIX FAST*.

[36] Qiao Li, Liang Shi, Chun Jason Xue, Kaijie Wu, Cheng Ji, Qingfeng Zhuge, and Edwin H.-M. Sha. 2016. Access Characteristic Guided Read and Write Cost Regulation for Performance Improvement on Flash Memory. In *USENIX FAST*.

[37] Tian Luo, Siyuan Ma, Rubao Lee, Xiaodong Zhang, Deng Liu, and Li Zhou. 2013. S-CAVE: Effective SSD Caching to Improve Virtual Machine Storage Performance. In *PACT*.

[38] Richard T. B. Ma, Dah Ming Chiu, John C. S. Lui, Vishal Misra, and Dan Rubenstein. 2010. Internet Economics: The Use of Shapley Value for ISP Settlement. In *IEEE/ACM Transactions on Networking, Vol. 18, No. 3*.

[39] Fei Meng, Li Zhou, Xiaosong Ma, Sandeep Uttamchandani, and Deng Liu. 2014. vCacheShare: Automated Server Flash Cache Space Management in a Virtualization Environment. In *USENIX ATC*.

[40] Micron. 2015. NAND Flash Memory MT29F[128/256]G08E[B/F]EBB, MT29F[128/256]G08E[B/E]CBB, MT29F512G08EMCBB.

[41] Muthukumar Murugan and David Du. 2011. Rejuvenator: A Static Wear Leveling Algorithm for Flash Memory. In *MSST*.

[42] Mohsin Naeem. 2019. Expect Shortage Of SSDs As Many Taiwanese Producers Shift Towards The Niche Markets. In *Appuals.com*.

[43] Dushyanth Narayanan, Austin Donnelly, and Antony Rowstron. 2008. Write Off-Loading: Practical Power Management for Enterprise Storage. In *USENIX FAST*.

[44] Yuanjiang Ni, Ji Jiang, Dejun Jiang, Xiaosong Ma, Jin Xiong, and Yuangang Wang. 2016. S-RAC: SSD Friendly Caching for Data Center Workloads. In *SYSTOR*.

[45] Junpeng Niu, Jun Xu, and Lihua Xie. 2018. Hybrid Storage Systems: A Survey of Architectures and Algorithms. In *IEEE Access, Vol. 6*.

[46] Yongseok Oh, Jongmoo Choi, Donghee Lee, and Sam H. Noh. 2012. Caching Less for Better Performance: Balancing Cache Size and Update Cost of Flash Memory Cache in Hybrid Storage Systems. In *USENIX FAST*.

[47] Oracle. 2015. Cloud Computing In Financial Services. In *Oracle Cloud for Industries*.
[48] Jian Ouyang, Shiding Lin, Song Jiang, Zhenyu Hou, Yong Wang, and Yuanzheng Wang. 2014. SDF: Software-Defined Flash for Web-Scale Internet Storage Systems. In *ASPLOS*.
[49] Stan Park and Kai Shen. 2012. FIOS: A Fair, Efficient Flash I/O Scheduler. In *USENIX FAST*.
[50] Western Digital (Liam Parker). 2018. Optimizing SSDs for Multiple Tenancy Use. In *Flash Memory Summit*.
[51] Craig Robertson. 2016. Software-Defined Flash: TradeOffs of FTL, Open-Channel, and Cooperative Flash Management. In *Storage Developer Conference*.
[52] Samsung. 2017. Ultra-Low Latency with Samsung Z-NAND SSD. In *Samsung*.
[53] SanDisk. 2016. Improving Performance and Endurance with Overprovisioning.
[54] Lloyd S. Shapley. 1952. A Value for n-Person Games. In *RAND Corporation*.
[55] Kai Shen and Stan Park. 2013. FlashFQ: A Fair Queueing I/O Scheduler for Flash-Based SSDs. In *USENIX ATC*.
[56] Zhaoyan Shen, Feng Chen, Yichen Jia, and Zili Shao. 2017. DIDACache: A Deep Integration of Device and Application for Flash Based Key-Value Caching. In *USENIX FAST*.
[57] Weijie Shi, Chuan Wu, and Zongpeng Li. 2018. A Shapley-value Mechanism for Bandwidth On Demand between Datacenters. In *IEEE Transactions on Cloud Computing, Vol. 6, No. 1*.
[58] David Shue, Michael J. Freedman, and Anees Shaikh. 2012. Performance Isolation and Fairness for Multi-Tenant Cloud Storage. In *USENIX OSDI*.
[59] Kent Smith. 2013. Understanding SSD Over Provisioning. In *Flash Memory Summit*.
[60] Arash Tavakkol, Juan Gomez-Luna, Mohammad Sadrosadati, Saugata Ghose, and Onur Mutlu. 2018. MQSim: A Framework for Enabling Realistic Studies of Modern Multi-Queue SSD Devices. In *USENIX FAST*.
[61] Arash Tavakkol, Mohammad Sadrosadati, Saugata Ghose, Jeremie S. Kim, Yixin Luo, Yaohua Wang, Nika Mansouri Ghiasi, Lois Orosa, Juan Gomez-Luna, and Onur Mutlu. 2018. FLIN: Enabling Fairness and Enhancing Performance in Modern NVMe Solid State Drives. In *ISCA*.
[62] Hui Wang and Peter Varman. 2014. Balancing Fairness and Efficiency in Tiered Storage Systems with Bottleneck-Aware Allocation. In *USENIX FAST*.
[63] Guanying Wu and Xubin He. 2012. Delta-FTL: Improving SSD Lifetime via Exploiting Content Locality. In *EuroSys*.
[64] Gala Yadgar, Eitan Yaakobi, and Assaf Schuster. 2015. Write Once, Get 50% Free: Saving SSD Erase Costs Using WOM Codes. In *USENIX FAST*.
[65] Zhengyu Yang, Morteza Hoseinzadeh, Allen Andrews, Clay Mayers, David (Thomas) Evans, Rory (Thomas) Bolt, Janki Bhimani, Ningfang Mi, and Steven Swanson. 2017. AutoTiering: Automatic Data Placement Manager in Multi-Tier All-Flash Datacenter. In *IPCCC*.
[66] Zhihao Yao, Ioannis Papapanagiotou, and Rean Griffith. 2015. Serifos: Workload Consolidation and Load Balancing for SSD Based Cloud Storage Systems. In *CoRR, abs/1512.06432*.
[67] Ning Zhang, Junichi Tatemura, Jignesh Patel, and Hakan Hacigumus. 2014. Re-evaluating Designs for Multi-Tenant OLTP Workloads on SSD-based I/O Subsystems. In *SIGMOD*.
[68] Rui Zhang, Ramani Routray, David M. Eyers, David Chambliss, Prasenjit Sarkar, Douglas Willcocks, and Peter Pietzuch. 2011. IO Tetris: Deep Storage Consolidation for the Cloud via Fine-Grained Workload Analysis. In *CLOUD*.
[69] Xuebin Zhang, Jiangpeng Li, Hao Wang, Kai Zhao, and Tong Zhang. 2016. Reducing Solid-State Storage Device Write Stress through Opportunistic In-place Delta Compression. In *USENIX FAST*.