Multi-Resource Fair Allocation for Consolidated Flash-Based Caching Systems

Wonil Choi
wonilchoi@hanyang.ac.kr
Hanyang University
Ansan, Republic of Korea

Bhuvan Urgaonkar
buu1@psu.edu
Pennsylvania State University
University Park, PA, USA

Mahmut Taylan Kandemir
mtk2@psu.edu
Pennsylvania State University
University Park, PA, USA

George Kesidis
gik2@psu.edu
Pennsylvania State University
University Park, PA, USA

ABSTRACT

Using a flash-based layer to serve the caching and buffering needs of multiple workloads has become a common practice. In such settings, resource demands will inevitably exceed available capacity sometimes. "Fair" resource allocation may offer a systematic way of partitioning resources across competing workloads during such periods of scarcity. Existing works only offer fair allocation strategies for a single resource (capacity or bandwidth) within a flash device in isolation. However, since there exist multiple critical resources that need to be partitioned within a flash device and they are correlated to each other, fair allocation of a single resource may result in a waste of other resource(s) or performance degradation of workload(s). To this end, we make a case for multi-resource fair allocation solutions for flash-based caches that consolidate multiple workloads. Furthermore, we argue that device lifetime, which depends on the behavior of running workloads, should also be considered as a first-class resource on par with capacity and bandwidth. Specifically, we build upon existing ideas related to dominant resource fairness (DRF) to devise flash-specific multi-resource fair algorithms: (i) nDRF, that jointly allocates capacity and bandwidth taking their non-linear relationship into account; (ii) ℓDRF, that explicitly considers lifetime as well in its allocation; and (iii) several variants of these. Our experimental evaluation offers important findings: (i) both nDRF and ℓDRF result in superior performance fairness compared to the state-of-the-art techniques that partition capacity in isolation; (ii) ℓDRF additionally offers improved device "wear" behavior; and (iii) our algorithms combined with reasonable demand prediction work very well in online settings with workload dynamism and uncertainty.

Middleware '22, November 7–11, 2022, Quebec, QC, Canada

ACM ISBN 978-1-4503-9340-9/22/11…$15.00
https://doi.org/10.1145/3528535.3565245

CCS CONCEPTS

• **Information systems → Flash memory**; **Storage management**; • **Social and professional topics → Pricing and resource allocation**.

KEYWORDS

solid-state drives, resource allocation, flash device lifetime

ACM Reference Format:
Wonil Choi, Bhuvan Urgaonkar, Mahmut Taylan Kandemir, and George Kesidis. 2022. Multi-Resource Fair Allocation for Consolidated Flash-Based Caching Systems. In *23rd International Middleware Conference (Middleware '22), November 7–11, 2022, Quebec, QC, Canada.* ACM, New York, NY, USA, 14 pages. https://doi.org/10.1145/3528535.3565245

1 INTRODUCTION

Due to the proliferation of flash devices and their growing capacities, consolidating multiple workloads on a single flash device – the solid state drive (SSD) or flash array – is now a common practice. While such consolidation improves resource utilization and enables novel use-cases, it inevitably leads to more frequent occurrence of scenarios wherein the resource needs of users (demand) may exceed available capacity (supply). Three canonical approaches for resource allocation in situations with such supply-demand mismatch are: (a) the use of priorities, (b) the use of some notion of fairness, and (c) the maximization of aggregate performance (when fairness is not of interest). When priorities are identical, or for users (workloads) within the same priority class, notions of fairness provide principled ways for resource allocation under resource scarcity. This paper explores possible ways of *fairly* allocating resources within a flash device for consolidated workloads.

Just like a server or a router, a flash-based device is a *composite* resource in the sense of being made up of multiple *fundamental* or *primitive* resources; each can be separately allocated to users and affects their performance in its own idiosyncratic manner. Therefore, any consolidation strategy for a flash device must, in general, worry about partitioning *all* relevant primitive resources. A large body of work exists on systematically partitioning a flash device's primitive resources – most commonly, storage capacity and/or network bandwidth (between host machine(s) and the flash device) [4, 19, 23, 24, 28, 33, 36, 39, 40, 46, 50, 51]. However, these approaches have two key shortcomings: (i) they only consider primitive resources in isolation, and (ii) they assume that other resources,

which are out of their interest, are ample, although in reality such resources may be a bottleneck.

First, treating each resource in isolation may lead to unbalanced allocations, which may be wasteful or performance-degrading [9, 13, 17, 18, 21, 22, 58]. Allocating a composite resource is a fundamentally more complex problem than merely applying separate fair partitioning decisions to individual primitive resources, since a workload's needs for different resources are typically *correlated* to each other. We note that *Dominant Resource Fairness* (DRF) [18] offers a systematic definition and a solution to this *multi-resource fair allocation* problem, and has been followed by several enhancements [9, 13, 17, 21, 22, 58]. However, adapting DRF to our flash context is *not* straightforward since DRF assumes that a workload's "demand vector" (i.e., a vector where each element represents the workload's demand for one primitive resource) always has individual resources' demands in a fixed proportion (e.g., 1 additional CPU for every 2 additional GB of DRAM). This is *not* the case for workloads' demands for flash resources. In this paper, we identify the relationship between a workload's demands for different flash resources, and enhance the DRF approach to take such complex relationship into account.

Second, while prior work allows each of consolidated workloads to consume as much flash lifetime as it wants (in allocating capacity and/or bandwidth), we note that finite flash *lifetime* deserves to be treated as a first-class resource on par with capacity and bandwidth. In fact, device lifetime is one prime consideration when a flash device is employed in a caching layer and needs to co-locate multiple workloads [16, 54], since, in general, consolidated workloads collectively issue more device-external and internal writes than a single workload does. In such a context, a recent work [12] explores possible ways of fairly allocating flash lifetime to consolidated workloads; unfortunately, this work treats flash lifetime in isolation and ignores its relationship with capacity and bandwidth allocations to each workload. In this paper, we reveal how a workload's lifetime demand relates to its capacity and bandwidth allocation, and present a comprehensive approach.

Target Scenario and Approach: Given a flash device (in a caching layer) and a set of workloads (to be consolidated on the device), an administrator needs to allocate the three major resources – capacity, bandwidth, and lifetime – of the device to the consolidated workloads. In this context, an important problem is: *How can one make a fair allocation by taking the complex relationship between each workload's demands for these three different resources?* In this work, we demonstrate how DRF [18] can be adapted to accommodate flash-specific characteristics of workloads' resource demands. A few important points related to our problem setting can be summarized as follows:

• We consider flash being used for *read caching* and *write buffering* above a slower hard disk-based storage layer (Section 2.3). Our concerns and ideas related to multi-resource fairness hold for a flash-based secondary storage layer as well but in a limited form. Since such a storage layer must accommodate all write requests, the allocation of lifetime becomes moot and the multi-resource fair allocation only involves capacity and bandwidth.

• While there might be various possible ways of evaluating the fairness, as suggested in [12], we evaluate it as how close performance values (e.g., response times) of consolidated workloads are to one another. Since fixing the total amount of lifetime resource in a certain period of time (limiting the number of writes that can be serviced in the fast flash caching layer) results in performance degradation, it would be meaningful to evaluate the fairness of the distribution of the performance burden across the workloads (Section 2.4).

• To perform DRF *at runtime*, one needs a separate mechanism to *predict* bandwidth and lifetime demands under a given capacity. While similar mechanisms proposed in prior work can be used, we present a simple prediction technique (Section 5) to evaluate our DRF in online settings. We confirm that the superiority of our DRF demonstrated in offline settings is maintained in online settings as well, even though the accuracy of the prediction technique is not perfect.

Contributions and Findings: Our experimental study reveals that a workload's demands for different flash resources are related to each other in complex, *non-linear* ways. Given this, we enhance DRF and devise: (a) non-linearity aware DRF (*n*DRF) that jointly allocates both capacity and bandwidth without considering lifetime, (b) lifetime-aware DRF (ℓDRF) that explicitly considers lifetime as well, and (c) several "baselines" that are representative of the state of the art. Following this, we propose a runtime framework that short-term predicts workloads' resource demands, computes suitable fair resource allocations for the workloads, and enforces these allocations using runtime mechanisms. An implementation of our proposed solution would span the flash device (capacity and bandwidth allocations) and the operating system (lifetime allocation and write control); see Section 5.3 for more details of our implementation. We perform extensive experimental evaluations using real-world workloads. Our key findings are as follows:

• The baselines or the lifetime-unaware *n*DRF can cause up to 23% more writes than our lifetime-aware ℓDRF.

• ℓDRF outperforms other strategies with write management in providing *performance fairness* (specifically, more equitable response times among workloads) for a variety of consolidation settings.

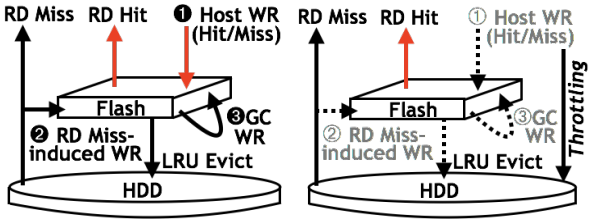
• In online settings, where near-term resource demands of workloads can be predicted, ℓDRF continues to provide superior performance fairness. Prediction errors may lead to an increase in the number of writes that may not be serviced by flash caching layer; however, this increase is much smaller (up to 5×) with ℓDRF than with the alternatives.

2 BACKGROUND AND RELATED WORK

2.1 Flash Device Basics

A flash device includes many blocks (the basic erase units); each block is divided into pages (the basic read/write units).

Garbage Collection: Since data over-writing is not allowed, flash employs an out-of-place update policy: when there is a data update (write), the old version of data is marked as *invalid*, and the new version of data is written into a clean page. This process gradually decreases the number of clean pages, and as a result, there is a need for reclaiming clean space to continue to serve incoming writes. Towards this, *garbage collection* (GC) picks so called victim blocks, moves valid pages in them to clean pages, and then erases them, thereby rendering the pages in these erased blocks clean and ready for writes.



(a) Normal Mode (b) Throttling Activation Mode

Figure 1: Flash/HDD access traffic in our system. RD: read; WR: write; HDD: hard disk drive; LRU: least-recently used.

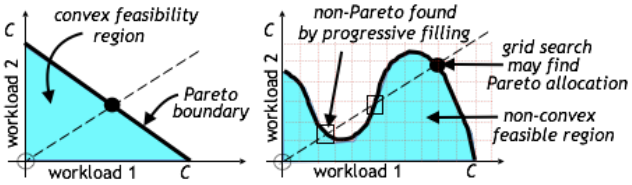


Figure 2: An allocation example by progressive filling (along dotted diagonal line) for two workloads/users sharing a single congested resource with an available total amount C.

Limited Lifetime: Flash cells' ability to secure data error-free deteriorates as write and erase operations are repeated. As a result, vendors specify the lifetime of their products as the number of writes/erases a device can endure. As is a well-regarded practice, we use the number of page writes remaining (or performed) as a *proxy* for remaining lifetime (or consumed lifetime). Also, since we view writes as a first-class resource, we use the phrases "consumed or allocated writes" analogous to "consumed bandwidth," "allocated capacity," etc. Among various write contributors (wear-leveling [8, 38], parity update [32, 37], refresh [7, 35], etc.), we focus on (i) *host writes* (directly issued by workloads) and (ii) *GC writes* (valid page movements during GC), since they may be neither optional nor deferrable, unlike wear-leveling and data-refresh.

2.2 Lifetime Management Knob

If a flash device needs to last for a desired amount of time (i.e., has a desired lifetime), it needs to be able to control the number and intensity of writes that occur on the device during appropriate periods of times. A related work [12] refers to such a period of write control as an "epoch", and the limit on the overall writes itself as the "write budget" for that epoch. Many reasonable policies are possible for determining these budgets and epoch lengths. Generally, both are time-varying quantities that can be adapted *dynamically*, to realize the desirable performance and lifetime behaviors. We adopt the same concepts for lifetime management in our work, but propose a new lifetime allocation strategy in such a context.

2.3 Scope of Our Work

Storage System Targeted: Flash devices are widely employed as part of a read-cache and write-buffer layer [14, 44, 48]. We target flash devices in such layers. In this paper, we consider the two-layer system shown in Figure 1: a flash device as a cache/buffer (henceforth, we simply use the term *cache*, with the understanding that the device will be used for both caching reads and buffering writes), with a magnetic hard disk drive (HDD) based persistent storage system in the next layer. Detailed cache operations and traffic across this hierarchy will be described later in Section 3.1. A key point to highlight is that this system employs a *write throttling* mechanism, which is our *knob* to control the consumption of flash lifetime. Throttled writes are directed to the HDD, and, therefore, experience longer response times, while being prevented from contributing to flash wear. In this work, we assume that bandwidth of the HDD is not a bottleneck as it will serve fewer reads and writes compared to the case where the caching layer is absent.

Flash Device Functionality Assumed: We assume a *soft-partitioned SSD*[1] – an existing, popular consolidated flash design [10, 23, 24, 41]. The key capabilities our target soft-partitioned SSD possesses are: (i) workload-awareness (given an I/O request, the device is able to identify which workload generated it), (ii) per-block workload ownership (the granularity of capacity sharing is a single block, though we do allow a given physical block to be reallocated over time), and (iii) per-workload GC invocation (each workload is responsible for invoking GC using its allocated user and OP blocks[2]). With these capabilities, it becomes possible to limit a workload to use only a specific set of flash blocks at a given time. Also, the number of writes due to each workload can be accurately determined. Note that these capabilities are already offered in some devices on the market (e.g., Samsung's multi-stream SSDs [10]).

2.4 DRF and Flash

Dominant resource fairness (DRF) [18] is an attractive starting point for us. DRF seeks an allocation that *equalizes* users' "dominant shares" and, in doing so, offers several desirable fairness properties. These properties are: (i) incentive compatibility, (ii) strategy-proofness, (iii) envy-freeness, and (iv) Pareto efficiency; see [18] for precise definitions of these properties and why/when they might be desirable. A user's dominant share is the maximum among her fractional needs for different resources. Concretely, given m different resources, let C_k be the total capacity of resource k and $d_{i,k}$ be user i's need for resource k to serve a user-specific unit of work (e.g., a job). Then, user i's dominant share is $s_i = \max_{k=1}^{m}\{d_{i,k}/C_k\}$.

Why We Can't Simply Use DRF As Is: Adapting DRF to our flash context is *not* straightforward as DRF assumes that user i's demands for different resources are always in the same proportion (say, $d_{i,k}/d_{i,t}$, for resources k and t), regardless of the total resources being procured. We observe from our experimental study that a user's demand for a flash resource tends to have a complex *non-linear* relationship with its demand for other resources. We implement a flash cache based on the least-recently used (LRU)

[1] Soft-partitioned SSDs allocate their capacity at a flash block granularity, while hard-partitioned SSDs do so at a coarser granularity (channel or chip). Generally, soft-partitioned SSDs are able to provide superior capacity utilization and degree of consolidation compared to their hard-partitioned counterparts [20, 30], but are prone to poor performance isolation.

[2] Flash devices typically contain more capacity than the number of user-perceived blocks. The main reason for this over-provisioned (OP) capacity is to relieve the high GC overhead. In general, the more OP capacity a flash device has, the fewer valid pages are moved during GC, and one consumes less bandwidth and needs fewer page writes.

replacement policy using the Disksim simulator. As Figure 3 shows, the lifetime demands of workloads are not linear functions of the capacity allocations; in fact, they may not even be convex/concave. This is because the flash writes due to a workload (its "write consumption") have two contributors, namely, *read-miss induced writes* and *GC induced writes*, both varying significantly with capacity allocations. Figure 4 shows that the bandwidth demands of workloads are also *not* linear functions of capacity; the bandwidth demand of a workload is determined by its *read hit ratio*, which depends on the allocated capacity.

Why We Need A New Allocation Mechanism: Assuming that there are m resource types and each workload (user) i has a resource demand vector $\underline{d}_i = (d_{i,1}, ..., d_{i,m})$, DRF can be realized via "progressive filling" (PF), where each active workload i is iteratively allocated a small increment $\varepsilon \underline{d}_i$ until one of the resources is exhausted. Due to the convexity of the set of feasible allocations that holds for the DRF formulation in [18], it is guaranteed that the allocation will be Pareto (i.e., no other allocation will give increased benefit to *all* of the workloads). But, if the set of feasible allocations is non-convex, it is possible that progressive filling will lead to non-Pareto allocations (see Figure 2 for an illustrative example, where we only show one of the n resources and 2 workloads).

Assessment of Fairness: While vanilla DRF provides the four fairness properties mentioned above, our DRF-inspired allocation schemes may not provably offer such fairness properties, since our work is based upon more *realistic demand* modeling. To our knowledge, most existing DRF-related works assume *rigid demand vectors* and use the traditional PF algorithm to prove the fairness properties. Instead, using a systematically-varied broad spectrum of workloads, we empirically and heuristically explore how our DRF-variants compare with state-of-the-art baselines in their ability to offer equitable performance. Specifically, as suggested in [12], given that imposing a write budget results in performance degradation, we evaluate the fairness of the distribution of the performance burden across the consolidated workloads, and thus, how equitable response times of the consolidated workloads are. Actually, our empirical study demonstrates that our strategies can deliver more equitable response times across consolidated workloads than the state-of-the-art mechanisms. One may consider a related work [59] which devises a user-specific utility function (Cobb-Douglas production function) to estimate resultant performance under the allocation of multiple resources, and examines whether it satisfies the four fairness properties. However, such an approach is not applicable to our problem, since the relationships of resource demands for different flash resources are *non-convex/concave* in general.

2.5 Related Work

Flash Capacity Partitioning: There is a large body of work [2, 4, 15, 28, 33, 34, 36, 39] for consolidated flash capacity management. A common high-level idea across these is to accurately identify the working set of each workload to allow judicious use of the limited flash capacity, and hence, save the limited flash capacity for other workloads while maintaining high hit ratios. However, *none of these works accounts for multi-resource fairness or lifetime as a first-class resource,* thus assuming that other resources such as bandwidth and lifetime are neither the bottleneck nor in need of management. We consider several allocation schemes representative of the state of the art in Section 4.4, and compare our proposal against them in Section 6.

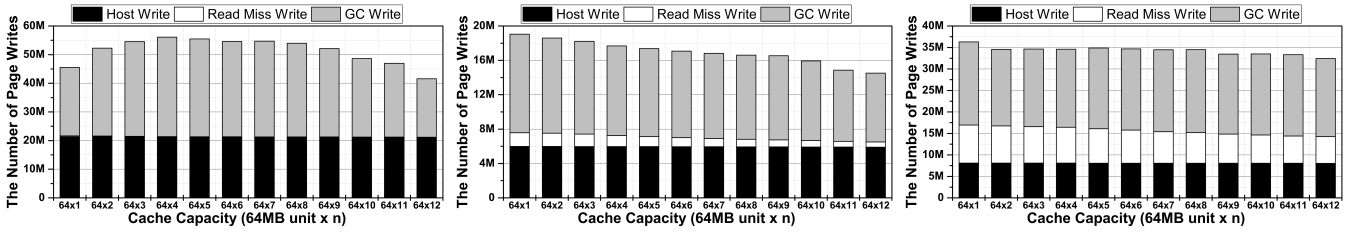
Flash Bandwidth Partitioning: Another large body of work [19, 40, 43, 46, 47, 50, 51] proposes techniques to provide a certain performance service level to each workload. Instead of explicitly partitioning flash-internal resources, they propose to re-schedule I/O requests at various levels of queues towards increasing their fairness metric values. Unfortunately, *this group of work assumes that flash resources such as capacity and lifetime are neither scarce nor bottleneck, which may not work when a resource becomes the bottleneck.* In contrast, our approach explicitly partitions various flash-internal resources that relate to each other, which can be still effective under resource scarcity.

DRF-based Multi-Resource Fair Allocation: DRF has been employed or enhanced to solve multi-resource fair allocation problems that exist in various domains [9, 13, 17, 18, 21, 22, 47, 58]. The multiple resources targeted by such works include {CPU, DRAM} in clusters/data centers[9, 18], {CPU, DRAM, network bandwidth} in heterogeneous servers [22, 58], bandwidths of multiple links in cloud [13, 17], bandwidths of different storage types in cloud [47], and capacities of different types of memories [21]. However, to our knowledge, *there has been no attempt to adapt DRF in the flash context.* We identify the multi-resource fair allocation problem in flash and revise DRF to take flash-specific characteristics into account.

DRF for Flash: DRF has been employed in problems where flash devices are involved [55–57]. However, *these problems merely treat a flash device (its bandwidth) as a resource in heterogeneous or hybrid storage systems where other storage types are involved as other resources.* In contrast, our main interest is on multiple different resources *within* a flash device. One particular work to note [11] uses vanilla DRF in the flash context, but for artificially-created scenarios where resource demands are neither correlated nor non-linear. In contrast, we develop practical DRF algorithms for more realistic settings that embrace such correlation and non-linearity.

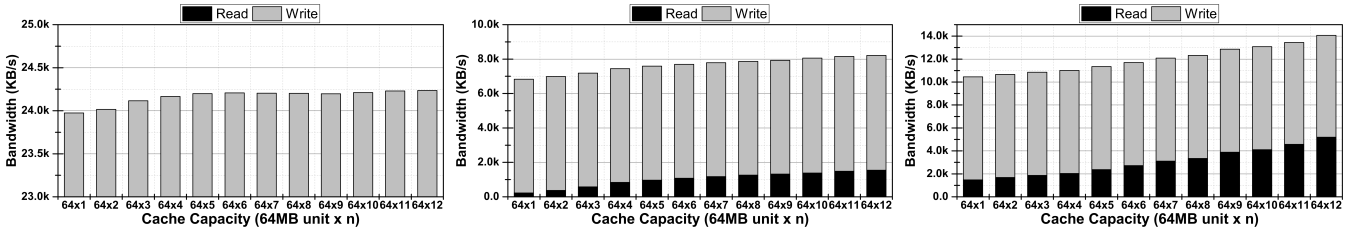
Flash Lifetime Allocation: There has been a recent work that treats flash lifetime as a resource to be partitioned across consolidated workloads [12]. This work explores various ways of fairly allocating a fixed number of writes to each workload. However, *the said work formulates a "single-resource" problem by treating flash lifetime in isolation, which is based on the assumption that other related resources are unlimited or not bottleneck.* We remove this assumption and consider all the relevant resources, lifetime, capacity, and bandwidth, as a composite resource to be allocated at a time.

Non-linearity in Resource Demands: Many of the existing resource allocation techniques model the potential benefits under different allocations to find an optimal allocation, which are generally non-linear functions. For example, utility functions (reduction in cache misses vs cache sizes) used in a CPU cache partitioning work [45], cacheability functions (read hit ratios vs cache sizes) developed in a flash cache partitioning technique [3], and cost functions (tail latencies vs IOPS tokens) leveraged in a local/remote flash access management technique [27] are all non-linear. In contrast, *the non-linearity discussed in our work indicates the one observed in relationship of a workload's demands for different resources (e.g., lifetime vs capacity), which is exploited by DRF.*



(a) $prxy$ **(b)** hm **(c)** prn

Figure 3: The total number of (page) writes three representative workloads consume under varying (SSD) cache capacities.



(a) $prxy$ **(b)** hm **(c)** prn

Figure 4: The bandwidth (KB/s) three representative workloads consume under varying (SSD) cache capacities.

3 CHARACTERIZING FLASH RESOURCE DEMANDS

We begin by characterizing how different workloads use different primitive resources – capacity, bandwidth, and lifetime – on flash. We are particularly interested in how, for different workloads, their need for one primitive resource is related to their need for the others and how these relationships are different from what DRF assumes. To investigate the amount of resources that would be *demanded (needed)* by a workload, we measure the amount of resources *consumed* by the workload when executed in an *unconsolidated* setting (i.e., with plentiful resources). We model bandwidth and lifetime demands as functions of capacity.

3.1 Cache and Buffer Mechanisms Assumed

Figure 1a illustrates traffic to/from our flash cache. Although we employ a simple cache mechanism based on LRU replacement, our approach can be generalized to other replacement policies as well. We disregard cold (read or write) misses.

• **Write:** In our experiments, all writes are serviced from the flash layer (i.e., no write budgets are enforced). If a write invalidates an old version but does not cause an LRU eviction, we label it a "write hit." On the other hand, a "write miss" invalidates and evicts the LRU data.

• **Read:** If the read data is found in the cache (hit), the read gets serviced by the flash cache; otherwise (miss), it gets serviced by the HDD, and the data is admitted into the cache by evicting the LRU data.

3.2 Simulation Methodology

We use the DiskSim [6] simulator with SSD extensions [1]. While there exist various possible ways of allocating storage capacity to a workload, we set a unit capacity and increase the total capacity by gradually allocating more unit capacities. In this experiments, we set the unit capacity to 64MB (we choose this granularity based on our empirical observations indicating that allocations at a finer granularity do not have any significant impact on the nature of our decision-making). We measure the number of page writes and the amount of bandwidth used, by varying the capacity allocation from 64MB (1 unit) to 768MB (12 units). We assume that 10% of the allocated capacity is reserved as the OP capacity, with the remaining 90% being the user-perceived/accessible space that can be used for caching/buffering. We execute 8 I/O traces from [31]. We present results of three representative workloads – *prxy*, *hm*, and *prn* – since resource demand patterns of the remaining workloads are quite similar to one of these three.

3.3 Capacity vs Lifetime Demand Analysis

Figure 3 plots the number of page writes for each workload under different capacity allocations from 1 to 12 units.

Contributors of Write Consumption: There are three sources that contribute to the total number of writes (refer to the traffic heading to flash device in Figure 1a). ❶ Host writes directly issued by a workload are a major contributor. ❷ Each read miss leads to a write for admitting the read data from HDD to flash; we call this a *read-miss induced write*. ❸ The flash internally triggers GC to secure clean space for continuing the write service, which is another contributor.

Observations: We make the following observations:

• The total number of writes eventually decreases, as the capacity allocation increases. This is because both the read-miss induced writes (❷) and GC writes (❸) decrease with the increasing capacity. Note also that the number of host writes (❶) is workload-specific and is not affected by any change in capacity. Specifically, the larger

Table 1: Comparison of our four DRF-based resource allocation strategies and four non-DRF strategies.

	Strategy	Resource Considered in Allocation			Lifetime Management		Objective of Resource Allocation
		Capacity	Bandwidth	Lifetime	Throttling	Automatic	
DRF-based	**nDRF (§4.1)**	✔	✔				$\forall i,j \;\; s_i = s_j \;\; such\ that \;\; s_i = \max(d_{i,k}/R_k) \;\; \forall k$, • i, j: workload number, • R_k: the total amount of resource k • $d_{i,k}$ is workload i's demand on resource k
	***n*DRF+Even (§4.2)**	✔	✔		✔		
	***n*DRF+MMF (§4.2)**	✔	✔		✔		
	ℓDRF (or ℓDRF+MMF) (§4.3)	✔	✔	✔		✔	
Non-DRF	**EqualHR (§4.4)**	✔					$\forall i,j \;\; maximize\ h_i \;\; \& \;\; h_i = h_j \;\; s.t. \;\; \sum c_i \le C$, • h_i: i's hit ratio • c_i: i's capacity • C: total capacity
	EqualHR+MMF (§4.4)	✔			✔		
	MaxCumHR (§4.4)	✔					$maximize \;\; \sum h_i \;\; subject\ to \;\; \sum c_i \le C$, • h_i: i's hit ratio • c_i: i's capacity • C: total capacity
	MaxCumHR+MMF (§4.4)	✔			✔		

the capacity, the lower the read miss ratio (and fewer the read-miss induced writes). Also, the larger the total capacity, the larger the OP capacity (10% of the total capacity), and the fewer the GC writes.

• Interestingly, one can observe from Figure 3a that the number of GC writes (and the total number of writes) increases as cache capacity increases during certain regions. According to our analysis, when cache capacity is very small, the newly-admitted data (which are written into a block in order) are not referenced until they become the LRU data and invalidated for eviction. Consequently, when GC is invoked, most victim blocks include few valid pages to move, which in turn significantly reduces the number of GC writes.

• The write (lifetime) demands of the workloads on flash cache are *not* linearly related to their capacity allocations. In fact, they may not even be convex/concave. This is because the number of both read-miss induced writes and GC writes decreases at a non-uniform rate as the cache capacity increases.

3.4 Capacity vs Bandwidth Demand Analysis

We are interested in the bandwidth of the interconnect between the host and our flash cache, which has been reported to be a common bottleneck [49]. This bandwidth is used to serve host read hits and host write hits/misses (see the red arrows in Figure 1a). Figure 4 plots the bandwidth consumed by each workload under different capacity allocations.

Observations: We make the following main observations.

• Across all workloads, the bandwidth consumption increases or saturates as the capacity allocation increases. This is because, the read miss ratios decrease and get closer to zero, and eventually all reads get serviced from the flash cache. Note that all writes get serviced by the flash regardless of hit/miss (which has a constant impact on the bandwidth consumption). Consequently, their contribution to bandwidth consumption is independent of capacity.

• The bandwidth demands of workloads are also *not* linear functions of capacity. This is because, as discussed above, the bandwidth consumption of a workload is determined by its read hit ratio, which depends on the allocated capacity.

4 PROPOSED RESOURCE ALLOCATION STRATEGIES

We propose and evaluate four DRF-inspired strategies, and compare them against four non-DRF strategies (ideal flash partitioning works in offline settings), summarized in Table 1.

4.1 Non-Linearity Aware DRF (*n*DRF)

*n*DRF considers *only capacity and bandwidth* in its decision-making and ignores lifetime. *n*DRF allocates the two different resource types, namely, capacity and bandwidth, *simultaneously*. It modifies the conventional PF as follows: it progressively allocates a unit capacity increment[3] and its corresponding bandwidth increment (determined by the gradient of bandwidth vs. capacity relationship at the currently allocated capacity), instead of fixed capacity and bandwidth increments. This iterative process continues until one of the two resources is fully allocated. The resource demand relationships are empirically profiled in offline settings (Section 3.4) and predicted in online settings (Section 5.2).

4.2 *n*DRF with Write Throttling

A natural way of adding lifetime management to *n*DRF is to prescribe a *write budget* for an epoch and employ *write throttling*, i.e., not using flash for servicing write requests in excess of the current epoch's budget. We explore two throttling mechanisms that can be combined with *n*DRF.

Even Throttling (*n*DRF+Even): We evenly divide the write budget across consolidated workloads, and permit each workload to consume only its allocated budget.

Max-Min Fair Throttling (*n*DRF+MMF): To allow for a fairer allocation of the write budget, we propose to employ the notion of max-min fairness (MMF) in dividing the total budget. Based on the estimated write demand of each workload under the *n*DRF allocation, we first attempt to allocate the number of writes that are demanded by the workload in need of the fewest writes to all the workloads. We then repeat this process, excluding the already-allocated writes, until the total write budget is fully allocated. In this manner, the writes that are not used by workloads in need of fewer writes under *n*DRF+Even can be used by the other workloads that are in need of more writes – *n*DRF+Even fails to offer this behavior. Note that, if the total write demand is sufficiently small, *n*DRF+MMF would be the same as *n*DRF+Even.

4.3 Lifetime-Aware *n*DRF (ℓDRF)

By considering lifetime as an explicit resource, ℓDRF allocates the three resource types, namely, capacity, bandwidth, and lifetime, *simultaneously*. Since lifetime vs capacity relationship may be non-convex/concave, a modified PF algorithm (as for *n*DRF) may work

[3]The suitable unit capacity depends on the specific consolidation setting (cache size, the number of workloads, etc). We set it to 64MB (Section 3.2), as using a finer granularity did not change resultant DRF allocations.

Table 2: nDRF vs ℓDRF allocations in an example consolidation scenario where three workloads are co-located.

Tot Rsc	**Cap:** 1,280MB, **BW:** 81,920 KB/s, **Write:** 100,000,000								
Strategy	n**DRF**				ℓ**DRF**				
Workload	**Cap(MB)**	**BW(KB/s)**	**Dominant Rrc**	**Dominant Share**	**Cap(MB)**	**BW(KB/s)**	**Write(#)**	**Dominant Rsc**	**Dominant Share**
$prxy$	640	24,180.30	Cap	0.500	64	23,973.71	45,538,103	Write	0.455
web	576	3,852.67	Cap	0.450	640	3,853.24	9,465,327	Cap	0.500
$proj$	64	42,278.53	BW	0.516	64	42,278.53	44,525,155	BW	0.516
Tot Alloc	1,280	70,311.50			768	70,105.48	99,528,585		

poorly (Figure 2). Instead, we employ a *grid search* to solve

$$\min_{\text{write allocations}} \sum_{i,j} (s_i - s_j)^2,$$

where s_i is the "dominant-resource share" of workload i, while exploring all possible allocations. An allocation for a workload consists of a multiple of the unit capacity and its corresponding bandwidth and writes. Any capacity partition under which the sum of resultant bandwidths or writes exceeds the total bandwidth or write budget, respectively, is *not* feasible and excluded from our consideration. Let t and n be the total number of capacity units in the device (to be partitioned) and the number of workloads; a fully-exhaustive grid-search evaluates a total of ${}^{(t-1)}C_{(n-1)}$ different allocations. We find that, if one employs a large capacity unit (and the number of possible capacity allocations is reduced), one may be able to perform an *exhaustive* grid search, and subsequently an exhaustive finer grid-search locally, all within a tolerable search latency.

Remarks: We discuss some noteworthy features of ℓDRF:

• *Non-Work Conservation:* ℓDRF is inherently not "work-conserving." That is, it may not allocate all available resources; thus, some workloads may not receive full service. This is a consequence of trying to *fairly* allocate lifetime; and the throttling approaches may not suffer from this issue as much. Further exploration of this trade-off between ℓDRF and nDRF+MMF is a part of our future work.

• *ℓDRF in Online Settings:* To perform ℓDRF in online settings, resource demands should be predicted for each workload, which may not be accurate, and hence, it also needs a throttling mechanism to prevent consolidated workloads from collectively consuming more writes than the budget. To this end, we evaluate ℓDRF+MMF in online settings (Section 6.3).

4.4 Non-DRF Baseline Strategies

We introduce four non-DRF allocation strategies that can be ideal in offline settings. State-of-the-art cache partitioning algorithms also aim the following goals in online settings.

Equalizing and Maximizing Hit Ratios (EqualHR): This strategy strives to equalize the hit ratios across the consolidated workloads while maximizing the aggregate hit ratio. We assume that bandwidth is not a bottleneck resource (i.e., collective bandwidth consumption ≤ total bandwidth).

EqualHR with MMF Throttling (EqualHR+MMF): Since Equal HR does not manage lifetime, one can employ write throttling as nDRF+Even or nDRF+MMF does (Section 4.2). Noting its superiority over the even partitioning of total write budget, this strategy uses the MMF partitioning as write throttling policy.

Maximizing Cumulative Hit Ratios (MaxCumHR): Ignoring fairness across the consolidated workloads, MaxCumHR seeks to maximize the sum of hit ratios of all workloads. Note that, in this case device lifetime is not considered.

MaxCumHR with MMF Throttling (MaxCumHR+MMF): To manage device lifetime while using MaxCumHR, this strategy employs write throttling based on the MMF division of the total write budget.

Table 3: Division of the total budget (100M) for write throttling. The shaded workloads experience throttled writes.

Strategy	$prxy$	web	$proj$
n**DRF (Actual Demand)**	53,635,289	9,526,917	44,525,155
n**DRF (Even Division)**	33,333,333	33,333,333	33,333,333
n**DRF (MMF Division)**	45,236,541	9,526,917	45,236,541
EqualHR (Actual Demand)	52,246,100	8,859,206	63,883,123
EqualHR (MMF Division)	45,570,397	8,859,206	45,570,397
MaxCumHR (Actual Demand)	52,246,100	8,859,206	63,883,123
MAxCumHR (MMF Division)	45,570,397	8,859,206	45,570,397

4.5 An Example Consolidation Scenario

We show how nDRF and ℓDRF allocate flash resources using an example workload consolidation scenario. We construct the scenario by combining $prxy$, web, and $proj$ from our workloads (Section 6.1). For our current discussion, it suffices to note that the device has a capacity of 1,280MB and a bandwidth to host 81,920KB/s. We choose a write budget of 100M over a week, which would allow the device to last about a year (if the write budget is fully utilized). For details on the device and how we choose the write budget, see Section 6.1. Table 2 gives the nDRF vs ℓDRF allocation results.

nDRF finds a resource allocation that equalizes dominant shares of the three workloads by considering only capacity and bandwidth (and ignoring write budget). Such an allocation consists of (640MB, 576MB, 64MB) capacity partition and the corresponding (24,180.30KB/s, 3,852.67KB/s, 42,278.53KB/s) bandwidth partition. Here, the dominant shares are (640MB/1,280MB=0.500, 576MB/1,280 MB=0.450, 42,278.53 KB/s/81,920KB/s=0.516), which are close to one another. The number of writes collectively consumed over a week by the three workloads under this nDRF allocation is 117,687,361, which is about 17.7% higher (more writes) than the given budget (100M).

ℓDRF offers a very different resource allocation. Whereas nDRF identifies the dominant resource of $prxy$ as capacity, ℓDRF regards its dominant resource as writes. The capacity allocation is (64MB, 640MB, 64MB).[4] The corresponding bandwidth and write allocations are (23,973.71KB/s, 3,853.24 KB/s, 42,278.53KB/s) and

[4]We chose this example to also illustrate how ℓDRF is not work-conserving (Section 4.3) – note that some capacity remains unallocated.

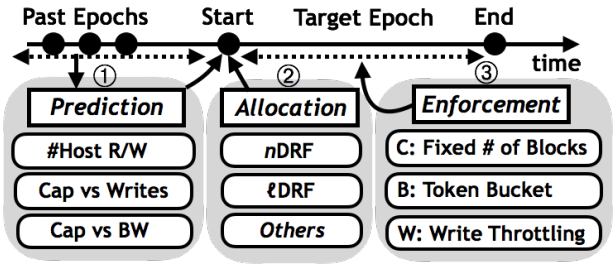


Figure 5: Our epoch-based resource allocation framework.

(45,538,103, 9,465,327, 44,525,155), respectively. The dominant shares (45,538,103/100,000,000=0. 455, 640MB/1,280MB=0.500, 42,278.53KB /s/81,920KB/s=0.516) are close to one another.

Both EqualHR and MaxCumHR result in a capacity allocation of (128MB, 768MB, 384MB), which is quite different from those of *n*DRF and *ℓ*DRF. This capacity allocation makes the three workloads collectively consume a total of 124,988,429 writes, which is 25% more than the given budget.

These strategies (in addition to *n*DRF) need to be combined with write *throttling* to manage lifetime. However, this write throttling has the potential to lead to unfairness across the workloads, as a result of the non-equitable degradation in response times as Section 6 will show. Table 3 lists the actual per-workload write demands vs Even/MMF partitioning of the 100M budget under the three allocation strategies. For each workload, further writes beyond those allocated are throttled (the shaded cells of the table). The MMF partitioning under *n*DRF is obviously better than the Even partitioning.

Vanilla DRF: One might want to see what vanilla DRF allocation would be for the same scenario. When assuming that resource demands of the three workloads are linearly-increasing, the capacity allocation is (512MB, 128MB, 128MB), which is entirely different from that of *ℓ*DRF. One cannot expect any benefit of DRF-like approaches, since such an allocation identifies wrong bottleneck resources.

5 ONLINE OPERATION

Workload consolidation scenarios can change *dynamically*; some workloads may terminate/leave, while others may start/join. Even for a given set of workloads, their resource demands can change over time. Under such realistic online settings, how can one treat different resources on an equal footing with one another, and allocate them to consolidated workloads? To this end, we present an online framework.

5.1 Epoch-based Resource Allocation Framework

We propose to make new resource allocation decisions at every "epoch," which is a period of *relative workload stationarity*. Upon a change in the workload consolidation scenario or any significant change in workloads' demand patterns, we end the current epoch and begin a new one.

Figure 5 illustrates how our framework works for each epoch. Its operation can be divided into three parts: ① (demand) prediction, ② (resource) allocation, and ③ (allocation) enforcement. At the beginning of a given epoch, ② our framework allocates the total amounts of capacity, bandwidth, and given write budget to all the consolidated workloads (Section 4). Here, note that, to make a DRF-based allocation, one needs to be aware of the resource demands (capacity vs bandwidth and capacity vs writes) of the workloads. To this end, ① our framework predicts workloads' resource demands by monitoring their patterns in the near-past epochs (Section 5.2). Once resources are allocated, ③ our framework enforces the allocation till the end of the epoch (Section 5.3).

5.2 Resource Demand Prediction

We first estimate the number of host reads and writes from the past workload execution, and then use them to estimate capacity vs bandwidth and capacity vs writes.

Predicting the Number of Host Requests: Among various possible predictors, we employ one that is *second-order autoregressive*, e.g., [29]. Assuming a uniform-sized monitoring interval (e.g., 1 minute), our predictor slides a window that monitors a set of the most recent past intervals (e.g., 10 intervals), and uses it to predict the # of host requests for the nearest future interval. Note that we use separate predictors for the # of host writes and the # of host reads, since they are independent from each other. Specifically, using a window whose size is W, the # of host requests for interval $n+1$, $\hat{x}(n+1)$, is obtained by:

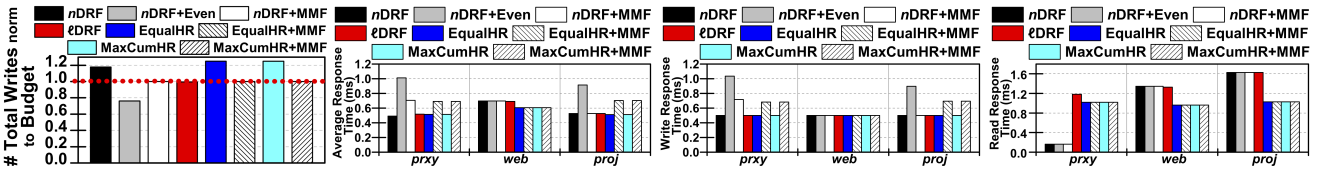
$$\hat{x}(n+1) \;=\; x_{avg} \;+\; b_n(x(n) - x_{avg}) \;+\; b_{n-1}(x(n-1) - x_{avg}),$$

where

$$x_{avg} = \textstyle\sum_{i=n-W+1}^{n} x(i) \;/\; W,$$

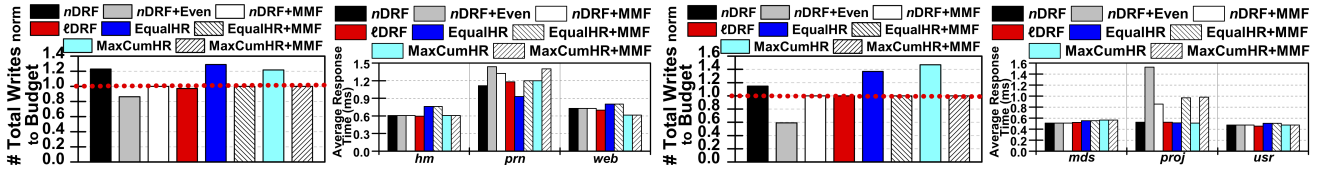
where $x(i)$ is the # of host requests monitored in interval i, and b^i is a weight representing the correlation between interval i and the predicted interval $n+1$, e.g., [29]. The accuracy of this predictor depends on various parameters, including the order of predictor, window size, and interval size. Motivated by our observation that the interval size significantly impacts the prediction accuracy (i.e., finer the interval, more accurate the prediction), we set the size of the interval to be much smaller than that of the epoch for which our framework makes resource-allocation decisions – e.g., epoch and interval can be set to 10 mins and 1 min, respectively (Section 6.3).

Predicting Capacity vs Writes and Capacity vs Bandwidth: We model the estimated total number of writes $\hat{y}$ as a function of the number of host writes (x_{hw}) and given capacity (c): $\hat{y} = f_w(x_{hw},\ c)$. Once we obtain such a model, we can estimate capacity vs writes, since we can predict the number of host writes ($\hat{x}_{hw}$), as discussed. Noting that we can collect a data sample like (x_i, c_i, y_i) for each of the past epochs, we cumulatively collect such data samples and apply *2D Lagrange Interpolation* [5] on them. Also, we model bandwidth ($\hat{z}$) as a function of the number of host reads (x_{hr}) and given capacity (c): $\hat{z} = f_b(x_{hr},\ c)$. Recall from Section 3.4 that a workload's bandwidth demand depends on the number of its host reads and its read hit ratio, whereas the number of its host writes has a constant impact. Using such a model, we can estimate capacity vs bandwidth, as we can predict the number of host reads ($\hat{x}_{hr}$). As in the case of capacity vs writes, we interpolate the data samples like (x_i, c_i, z_i) collected from the past epochs.



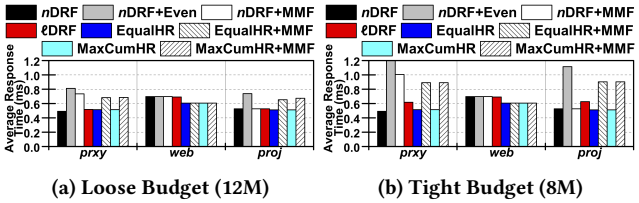
(a) # Consumed Writes (b) Average Response Time (ms) (c) Write Response Time (ms) (d) Read Response Time (ms)

Figure 6: Lifetime, and average, write, and read response times of consolidation scenario A under eight allocation strategies.



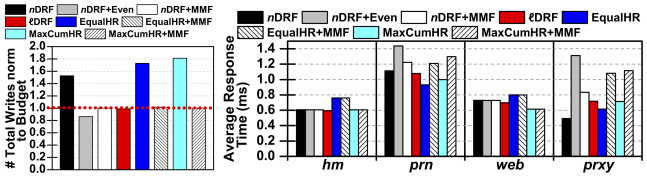
(a) (Consol-B) # Consumed Writes (b) (Consol-B) Response Time (c) (Consol-C) # Consumed Writes (d) (Consol-C) Response Time

Figure 7: Lifetime and average response times of other two representative consolidation scenarios B and C.



(a) Loose Budget (12M) (b) Tight Budget (8M)

Figure 8: Performance of scenario *A* under different budgets.



(a) # Total Writes (b) Response Times (ms)

Figure 9: A scenario where 4 workloads are consolidated.

5.3 Enforcement of Allocation during an Epoch

The three resources are managed in the following fashion:

Capacity Enforcement: Once a certain amount of capacity is allocated to a workload, that amount is *guaranteed* to the workload during the epoch. Specifically, the number of flash blocks, which a workload can use during the epoch, is fixed. Due to wear-leveling concerns, the physical blocks underlying a workload's allocation can vary over time (i.e., once a block that belongs to a workload is erased, it may belong to another workload).

Bandwidth Enforcement: To limit the amount of traffic a workload can generate during an epoch, we employ a *token bucket* mechanism [42, 53]. Specifically, while the amount of per-second bandwidth allocated to each workload is accumulated during the epoch (*tokens* are periodically added to a limited-sized *bucket*), the workload consumes the accumulated bandwidth (*tokens* are consumed).

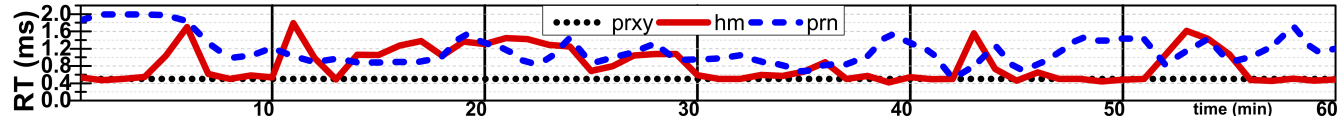
Write Budget Enforcement: To prevent a workload from consuming more than the allocated write budget, once the budget runs out, we remove two sources of writes (as described in Figure 1b). First, we stop serving ① host writes by employing *write throttling*, which redirects further host writes to the HDD in the next layer. Second, we give up ② admitting the data into the cache for read misses (i.e., read-miss induced writes), by just reading them from the HDD. Consequently, these two actions collectively prevent ③ GC (writes) from being generated any longer till the end of epoch.

6 EVALUATION

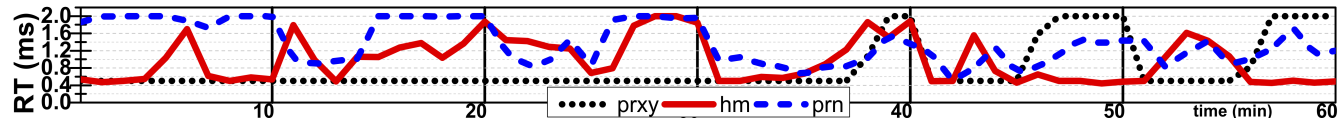
6.1 Experimental Setup

Simulation Framework: We implement the eight allocation strategies using MATLAB, which takes system's total resources and workloads' resource demands as inputs, and outputs a resource allocation. The resulting allocation is used by our framework, which is built using DiskSim [6] + SSD extension [1] simulator. Specifically, we implement a soft-partitioned SSD (Section 2.3) and add an LRU-based cache mechanism (Section 3.1) to use the SSD as a cache. We finally add a module that predicts resource demands (Section 5.2) and enforces an allocation (Section 5.3).

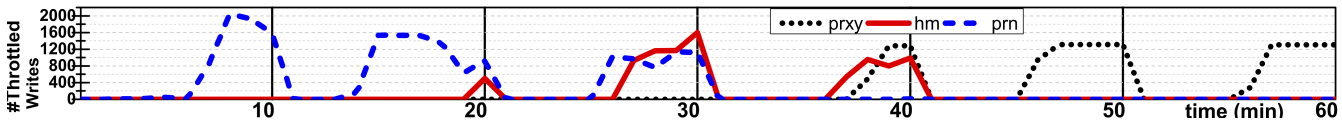
Device Configuration: While our framework is applicable to any device capacity and workload consolidation scenario, we reduce the problem size to be able to explore a large experimental space. Note that the multi-resource fair allocation problem exists in any situation where *demands exceed capacity*, regardless of the target flash capacity. We assume a 2GB SSD that includes 1,920 blocks, each consisting of 64 16KB pages. For the SSD latencies, we use 175us, 400us, and 3ms for read, write, and erase, respectively, from the datasheet of a modern Micron 3D flash SSD device [52]. As these values are measured at a drive level, we do not model a separate built-in DRAM within the SSD. For per-workload GC and system-wide wear-leveling, we employ, respectively, the widely-used greedy [25] and static [38] algorithms. For the HDD-based



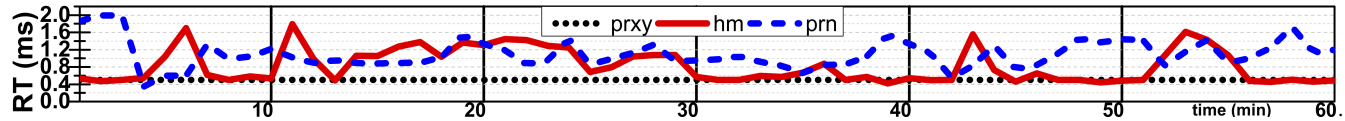
(a) The response times under *n*DRF over six consecutive 10-min epochs (1 hour).



(b) The response times under *n*DRF+MMF over six consecutive 10-min epochs (1 hour).



(c) The number of throttled writes under (b) *n*DRF+MMF over six consecutive 10-min epochs (1 hour).



(d) The response times under *ℓ*DRF over six consecutive 10-min epochs (1 hour)

Figure 10: The response times of 3 consolidated workloads under 3 different strategies, assuming *perfect* demand prediction.

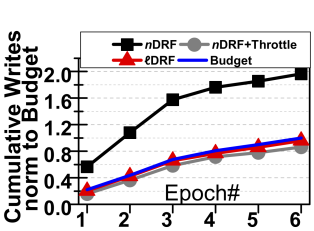
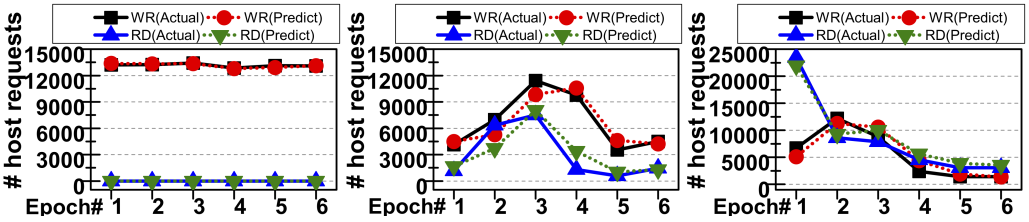


Figure 11: # writes under the 3 strategies over 6 epochs.



(a) prxy (b) hm (c) prn

Figure 12: Actual # vs predicted # of host requests over 6 epochs for three workloads.

layer, we use read/write latencies of 2ms [26]. We choose our write budgets as follows. We assume our SSD has 20K guaranteed P/E cycles – this amounts to a total of 3,000M page writes. Assuming we need to operate our SSD at least for a year, we distribute the 3,000M page writes the device can endure evenly over a year.[5] This amounts to 10M writes per day, and 60K writes over an epoch of 10 minute duration.

Workloads: We construct various consolidation scenarios by combining individual I/O traces from [31]. For online settings, to cope with changes in resource demands over time, we divide each of the I/O traces into 10-minute chunks (epochs).

Evaluated Strategies: We evaluate the four DRF-based and four non-DRF strategies, discussed earlier in Section 4.

Metrics: We are interested in the following two metrics:

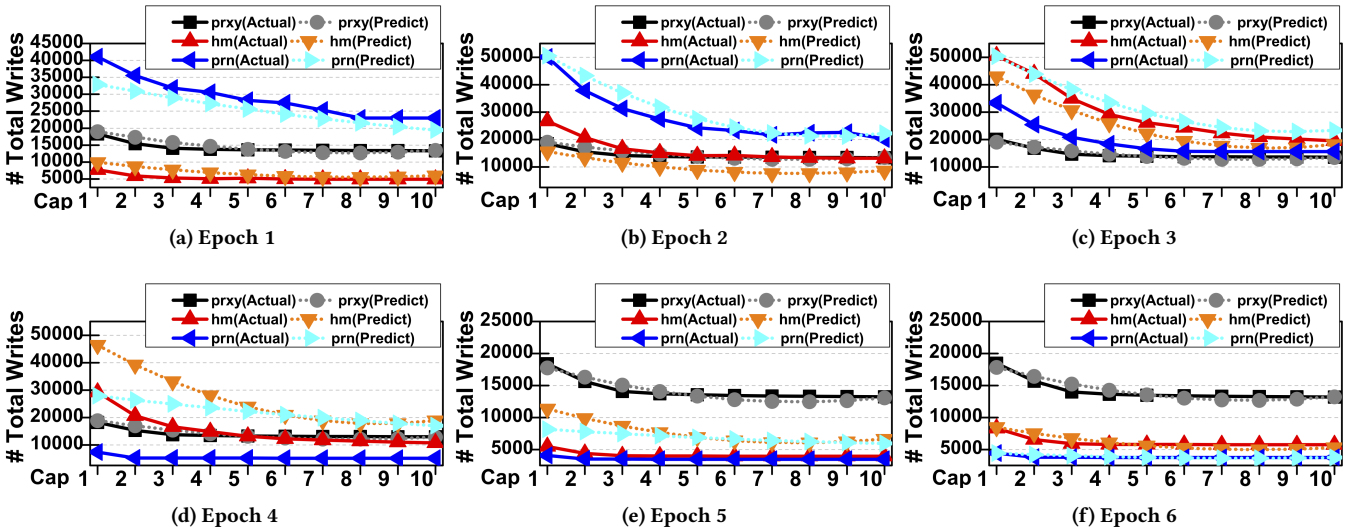
[5]Commonly used device refresh cycles are in the 1-3 year range. We leave more sophisticated budget selection for future work.

• **Lifetime Management:** Is adopting *ℓ*DRF or employing a write throttling for other strategies effective in suppressing the number of writes within a prescribed budget? How many writes do consolidated workloads collectively consume under the lifetime-unaware strategies?

• **Performance Fairness:** Given that imposing a write budget results in performance degradation, how can this burden be fairly distributed across the consolidated workloads? To measure this, we compare the response times of workloads.

6.2 Offline Setting Results

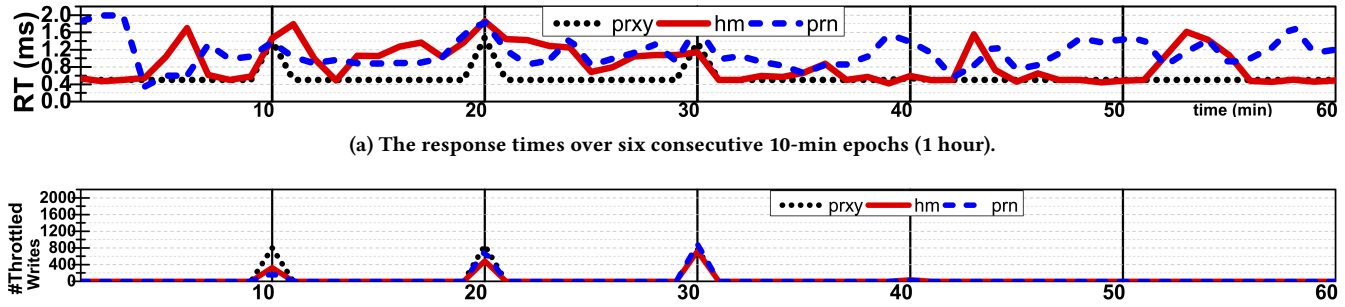
6.2.1 Lifetime Management Analysis. Figures 6a, 7a, and 7c show the number of writes consumed by three consolidation scenarios: $A(prxy, web, proj)$ used in Section 4.5, $B(hm, prn, web)$, and $C(mds, proj, usr)$. The results are *normalized* to the prescribed total budget (red line).



(a) Epoch 1 (b) Epoch 2 (c) Epoch 3

(d) Epoch 4 (e) Epoch 5 (f) Epoch 6

Figure 13: Actual vs predicted number of total writes under varying capacities for three workloads in each epoch.



(a) The response times over six consecutive 10-min epochs (1 hour).

(b) The number of throttled writes over six consecutive 10-min epochs (1 hour).

Figure 14: Three consolidated workloads under ℓDRF+MMF, based on our demand prediction.

• The three lifetime-unaware strategies without any write throttling, *n*DRF, EqualHR, and MaxCumHR, consume much more writes than the prescribed budget (up to 23%).
• Under the other five strategies, the number of consumed writes never goes beyond the budget. Specifically, strategies with a throttling that forces MMF division of the budget, namely, *n*DRF+MMF, EqualHR+MMF and MaxCumHR+MMF, fully consume the budget, whereas *n*DRF+Even and ℓDRF may leave many writes unused. This is because, for the former, some workloads that generate relatively fewer writes but are allocated evenly-divided budgets may consume fewer writes than their allocations allow; for the latter, its allocation aims at equalizing the dominant shares.
• To conclude, if the operator wants to manage her device's lifetime, she can impose a budget on the targeted workload consolidation, and adopt ℓDRF or employ write throttling.

6.2.2 Performance Fairness Analysis. Figures 6b, 6c, and 6d plot the average, write, and read response times, respectively, of the scenario $A(prxy, web, proj)$.

• Under the three strategies that do not manage lifetime (*n*DRF, EqualHR, and MaxCumHR), the response times of the three workloads are close to one another. The response time of *prxy* with high write-intensity remains relatively small, as all of its writes can get serviced from the flash.
• Among the other five strategies that keep the number of consumed writes from going over budget, ℓDRF *outperforms* the others with write throttling. This is because ℓDRF identifies *prxy* as a write resource-dominant workload, thereby allocating more writes to it than the others, while allocating more capacity or bandwidth to the others. Figure 6c reveals that, under ℓDRF, all writes of *prxy* get serviced from the flash.
• Figure 6d shows that the read performance of *prxy* significantly suffers under ℓDRF. This is because ℓDRF allocates a small capacity to it (and in turn, many of its reads miss in the flash cache, and get serviced from the HDD), while allocating more writes to it. Note however that, *prxy* is a write-dominant workload whose reads are

few, leading to a negligible impact on the overall response time (Figure 6b).
• We observe across a wide range of scenarios that ℓDRF is superior in providing performance fairness across workloads when a write constraint is imposed. For the two other consolidation scenarios, *B* and *C*, ℓDRF renders the response times of all workloads close to one another (see Figures 7b and 7d, respectively), while consuming only the prescribed write budget (see Figures 7a and 7c, respectively).

6.2.3 Budget Sensitivity Analysis. We have also evaluated the performance fairness provided by our strategies under varying budget sizes. Figures 8a and 8b plot the response times of the workloads in scenario *A* when 12M and 8M budgets are given, respectively. When the budget size is large (i.e., each workload is likely to consume as many writes as it wants), all lifetime management strategies provide a certain level of performance fairness. However, when the budget size is small (i.e., device lifetime is strictly managed for a longer use), ℓDRF is the best option to achieve high levels of performance fairness.

6.2.4 Degree of Consolidation Analysis. We evaluate our allocation strategies by varying the number of workloads in a consolidation set. As an example, we consider a new scenario where 4 workloads are consolidated by adding *prxy* to the scenario $B(hm, prn, web)$ used above. Figure 9a shows that the number of writes collectively consumed by the four workloads significantly increases (up to 81% more writes than the budget) under the lifetime-unaware strategies. When lifetime is considered, ℓDRF does an excellent job in providing performance fairness, while the results get worse under the write throttling-based strategies (Figure 9b).

6.3 Online Setting Results

Due to the space constraint, we present a representative scenario – *prxy*, *hm*, and *prn*, are combined – in which workloads' resource demands are quite different from one another and change significantly over time. The epoch is set to 10 mins and the budget for each epoch is set to 60K (Section 6.1). Focusing on six consecutive epochs (60 mins) where the workloads' demands change dynamically, we discuss our evaluation results under *n*DRF vs *n*DRF+MMF(Throttling) vs ℓDRF.

6.3.1 With Perfect Prediction. Assuming that our demand predictor (Section 5.2) is perfect, we execute our framework.
• Figure 11 shows the cumulative write consumption during the six epochs, *normalized* to the cumulative budget. The two lifetime-aware strategies, *n*DRF+MMF and ℓDRF, do not consume any writes beyond the budget, whereas *n*DRF consumes almost 2× more writes than the allowed budget.
• Figures 10a, 10b, and 10d plot the response times of the three workloads under *n*DRF, *n*DRF+MMF, and ℓDRF, respectively. ℓDRF provides a high level *performance fairness*, which is very close to that provided by *n*DRF. In contrast, one can observe that the gap between the response times becomes significantly large under *n*DRF+MMF. Figure 10c plots the number of throttled writes under *n*DRF+MMF, which causes an increase in response times.

6.3.2 Prediction Accuracy Analysis. We now evaluate our *near-term demand predictor*.
• Figures 12a, 12b, and 12c compare the actual vs predicted number of host writes and reads for *prxy*, *hm*, and *prn*, respectively. For *prxy* whose host I/O pattern does not change over epochs, our prediction is accurate. For *hm* and *prn*, where the host I/O patterns change dramatically over epochs, our prediction is quite accurate as well. With tolerable errors, our predictor can detect increasing or decreasing trends observed in the recent past, and quickly adapt it for the near future.
• Figure 13 shows the actual vs predicted number of total writes under varying capacities for the three workloads in each of the six epochs. The prediction results are very accurate for some epochs (e.g., epochs 1 and 5), while producing some errors in other epochs (e.g., epochs 3 and 4). Errors in the latter originate from the errors in the prediction of the number of host writes.

6.3.3 With Our Prediction. Considering such (tolerable) errors from our predictor, does our framework still work well?
• For each of the six epochs, *n*DRF allocation using our prediction is equal to that using the perfect prediction. This is because *n*DRF does not consider capacity vs write demand that may be error-prone when estimated.
• The performance fairness achieved by ℓDRF+MMF (Figure 14a) is not significantly different from that under ℓDRF with perfect prediction (Figure 10d), except that workloads experience slight write throttling in the last minutes of some epochs. Figure 14b shows such throttled writes. One can also conclude that, ℓDRF+MMF still *outperforms* *n*DRF+MMF, when comparing their number of throttled writes (Figure 14b vs Figure 10c).

7 CONCLUDING REMARKS

We made the case that fair resource allocation in a flash-based cache/buffer layer shared by multiple workloads is a novel and complex problem for two main reasons. First, a workload's capacity and bandwidth are related in a non-linear fashion, which existing theory on multi-resource fairness (DRF) does not accommodate. Second, flash has a finite lifetime which may be viewed as a first-class resource – doing so does not only improve device wear but also has implications on performance fairness. We devised *n*DRF and ℓDRF, enhancements to DRF, to capture these non-linearities. Our evaluations with a diverse set of real-world workloads demonstrated the consistent efficacy of our techniques over the state of the art.

ACKNOWLEDGMENTS

This work is supported in part by NSF grants 1931531, 2028929, 1931531, 2122155, and 1908793. This work is also supported by the MOTIE (Ministry of Trade, Industry & Energy) (1415181081) and KSRC (Korea Semiconductor Research Consortium) (20019402) support program for the development of the future semiconductor device, and Institute of Information & communications Technology Planning & Evaluation (IITP) grant funded by the Korea government (MSIT) (2022-0-00117). Wonil Choi is the corresponding author and supported by the research fund of Hanyang University (HY-2021-2596).

REFERENCES

[1] Nitin Agrawal, Vijayan Prabhakaran, Ted Wobber, John D. Davis, Mark Manasse, and Rina Panigrahy. 2008. Design Tradeoffs for SSD Performance. In *USENIX 2008 Annual Technical Conference* (Boston, MA, June 22 - 27, 2008) *(ATC '08)*. USENIX Association, USA, 57–70. https://dl.acm.org/doi/10.5555/1404014.1404019

[2] Saba Ahmadian, Onur Mutlu, and Hossein Asadi. 2018. ECI-Cache: A High-Endurance and Cost-Efficient I/O Caching Scheme for Virtualized Platforms. *Proc. ACM Meas. Anal. Comput. Syst.* 2, 1, Article 9 (April 2018), 34 pages. https://doi.org/10.1145/3179412

[3] Christoph Albrecht, Arif Merchant, Murray Stokely, Muhammad Waliji, François Labelle, Nate Coehlo, Xudong Shi, and C. Eric Schrock. 2013. Janus: Optimal Flash Provisioning for Cloud Storage Workloads. In *Proceedings of the 2013 USENIX Conference on Annual Technical Conference* (San Jose, CA, June 26 - 28, 2013) *(USENIX ATC'13)*. USENIX Association, USA, 91–102. https://dl.acm.org/doi/10.5555/2535461.2535473

[4] Dulcardo Arteaga, Jorge Cabrera, Jing Xu, Swaminathan Sundararaman, and Ming Zhao. 2016. CloudCache: On-Demand Flash Cache Management for Cloud Computing. In *Proceedings of the 14th Usenix Conference on File and Storage Technologies* (Santa Clara, CA, February 22 - 25, 2016) *(FAST '16)*. USENIX Association, USA, 355–369. https://dl.acm.org/doi/10.5555/2930583.2930610

[5] A. R. Bozorgmanesh, M. Otadi, A. A. Safe Kordi, F. Zabihi, and M. Barkhordari Ahmadi. 2009. Lagrange Two-Dimensional Interpolation Method for Modeling Nanoparticle Formation During RESS Process. *International Journal of Industrial Mathematics* 1, 2 (April 2009), 175–181. https://ijim.srbiau.ac.ir/article_2021.html

[6] John S. Bucy, Jiri Schindler, Steven W. Schlosser, and Gregory R. Ganger. 2008. *The DiskSim Simulation Environment Version 4.0 Reference Manual (CMU-PDL-08-101)*. Technical Report. Carnegie Mellon University.

[7] Yu Cai, Gulay Yalcin, Onur Mutlu, Erich F. Haratsch, Adrian Cristal, Osman S. Unsal, and Ken Mai. 2012. Flash correct-and-refresh: Retention-aware error management for increased flash memory lifetime. In *2012 IEEE 30th International Conference on Computer Design (ICCD)* (Montreal, QC, Canada, September 30 - October 3, 2012) *(ICCD '12)*. IEEE, USA, 94–101. https://doi.org/10.1109/ICCD.2012.6378623

[8] Yuan-Hao Chang, Jen-Wei Hsieh, and Tei-Wei Kuo. 2010. Improving Flash Wear-Leveling by Proactively Moving Static Data. *IEEE Trans. Comput.* 59, 1 (Jan. 2010), 53–65. https://doi.org/10.1109/TC.2009.134

[9] Wei Chen, Jia Rao, and Xiaobo Zhou. 2017. Preemptive, Low Latency Datacenter Scheduling via Lightweight Virtualization. In *Proceedings of the 2017 USENIX Conference on Usenix Annual Technical Conference* (Santa Clara, CA, July 12 - 14, 2017) *(USENIX ATC '17)*. USENIX Association, USA, 251–263. https://dl.acm.org/doi/abs/10.5555/3154690.3154714

[10] Changho Choi. 2017. *AutoStream: Automatic Stream Management for Multi-stream SSDs in Big Data Era.* Retrieved October 10, 2022 from https://www.snia.org/educational-library/autostream-automatic-stream-management-multi-stream-ssds-big-data-era-2017

[11] Wonil Choi, Bhuvan Urgaonkar, Mahmut Kandemir, and Myoungsoo Jung. 2019. Fair Resource Allocation in Consolidated Flash Systems. In *Proceedings of the 11th USENIX Conference on Hot Topics in Storage and File Systems* (Renton, WA, July 8 - 9, 2019) *(HotStorage '19)*. USENIX Association, USA, 22. https://dl.acm.org/doi/10.5555/3357062.3357091

[12] 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* (Lausanne, Switzerland, March 16 - 20, 2020) *(ASPLOS '20)*. ACM, USA, 1063–1076. https://doi.org/10.1145/3373376.3378502

[13] Mosharaf Chowdhury, Zhenhua Liu, Ali Ghodsi, and Ion Stoica. 2016. HUG: Multi-Resource Fairness for Correlated and Elastic Demands. In *Proceedings of the 13th USENIX Conference on Networked Systems Design and Implementation* (Santa Clara, CA, March 16 - 18, 2016) *(NSDI '16)*. USENIX Association, USA, 407–424. https://dl.acm.org/doi/10.5555/2930611.2930638

[14] Bertrand Dufrasne, In Kyu Park, Francesco Perillo, Hank Sautter, Stephen Solewin, and Anthony Vattathil. 2012. *Solid-State Drive Caching in the IBM XIV Storage System.* Retrieved October 10, 2022 from https://www.redbooks.ibm.com/redpapers/pdfs/redp4842.pdf

[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 *Proceedings of the 16th USENIX Conference on Networked Systems Design and Implementation* (Boston, MA, February 26 - 28, 2019) *(NSDI '19)*. USENIX Association, USA, 65–78. https://dl.acm.org/doi/10.5555/3323234.3323241

[16] Hemant Gaidhani. 2015. *Speeds, Feeds and Needs – Understanding SSD Endurance.* Retrieved October 10, 2022 from https://blog.westerndigital.com/ssd-endurance-speeds-feeds-needs/

[17] Ali Ghodsi, Vyas Sekar, Matei Zaharia, and Ion Stoica. 2012. Multi-Resource Fair Queueing for Packet Processing. In *Proceedings of the ACM SIGCOMM 2012 Conference on Applications, Technologies, Architectures, and Protocols for Computer Communication* (Helsinki, Finland, August 13 - 17, 2012) *(SIGCOMM '12)*. ACM, USA, 1–12. https://doi.org/10.1145/2342356.2342358

[18] Ali Ghodsi, Matei Zaharia, Benjamin Hindman, Andy Konwinski, Scott Shenker, and Ion Stoica. 2011. Dominant Resource Fairness: Fair Allocation of Multiple Resource Types. In *Proceedings of the 8th USENIX Conference on Networked Systems Design and Implementation* (Boston, MA, March 30 - April 1, 2011) *(NSDI '11)*. USENIX Association, USA, 323–336. https://dl.acm.org/doi/10.5555/1972457.1972490

[19] Mohammad Hedayati, Kai Shen, Michael L. Scott, and Mike Marty. 2019. Multi-Queue Fair Queueing. In *Proceedings of the 2019 USENIX Conference on Usenix Annual Technical Conference* (Renton, WA, July 10 - 12, 2019) *(USENIX ATC '19)*. USENIX Association, USA, 301–314. https://dl.acm.org/doi/10.5555/3358807.3358834

[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 *Proceedings of the 15th Usenix Conference on File and Storage Technologies* (Santa clara, CA, February 27 - March 2, 2017) *(FAST '17)*. USENIX Association, USA, 375–390. https://dl.acm.org/doi/10.5555/3129633.3129667

[21] Sudarsun Kannan, Ada Gavrilovska, Vishal Gupta, and Karsten Schwan. 2017. HeteroOS: OS Design for Heterogeneous Memory Management in Datacenter. In *Proceedings of the 44th Annual International Symposium on Computer Architecture* (Toronto, ON, Canada, June 24 - 28, 2017) *(ISCA '17)*. ACM, USA, 521–534. https://doi.org/10.1145/3079856.3080245

[22] Jalal Khamse-Ashari, Ioannis Lambadaris, George Kesidis, Bhuvan Urgaonkar, and Yiqiang Zhao. 2018. An Efficient and Fair Multi-Resource Allocation Mechanism for Heterogeneous Servers. *IEEE Transactions on Parallel and Distributed Systems* 29, 12 (Dec. 2018), 2686–2699. https://doi.org/10.1109/TPDS.2018.2841915

[23] Bryan S Kim. 2018. Utilitarian Performance Isolation in Shared SSDs. In *Proceedings of the 10th USENIX Conference on Hot Topics in Storage and File Systems* (Boston, MA, July 9 - 10, 2018) *(HotStorage '18)*. USENIX Association, USA, 16. https://dl.acm.org/doi/10.5555/3277332.3277348

[24] Jaeho Kim, Donghee Lee, and Sam H. Noh. 2015. Towards SLO Complying SSDs through OPS Isolation. In *Proceedings of the 13th USENIX Conference on File and Storage Technologies* (Santa Clara, CA, February 16 - 19, 2015) *(FAST '15)*. USENIX Association, USA, 183–189. https://dl.acm.org/doi/10.5555/2750482.2750496

[25] Jaeho Kim, Kwanghyun Lim, Young-Don Jung, Sungjin Lee, Changwoo Min, and Sam H. Noh. 2019. Alleviating Garbage Collection Interference through Spatial Separation in All Flash Arrays. In *Proceedings of the 2019 USENIX Conference on Usenix Annual Technical Conference* (Renton, WA, July 10 - 12, 2019) *(USENIX ATC '19)*. USENIX Association, USA, 799–812. https://dl.acm.org/doi/10.5555/3358807.3358875

[26] 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 *2011 IEEE 19th Annual International Symposium on Modelling, Analysis, and Simulation of Computer and Telecommunication Systems* (Singapore, July 25-27, 2011) *(MASCOTS '11)*. IEEE, USA, 227–236. https://doi.org/10.1109/MASCOTS.2011.64

[27] Ana Klimovic, Heiner Litz, and Christos Kozyrakis. 2017. ReFlex: Remote Flash Local Flash. In *Proceedings of the Twenty-Second International Conference on Architectural Support for Programming Languages and Operating Systems* (Xi'an, China, April 8 - 12, 2017) *(ASPLOS '17)*. ACM, USA, 345–359. https://doi.org/10.1145/3037697.3037732

[28] Ricardo Koller, Ali José Mashtizadeh, and Raju Rangaswami. 2015. Centaur: Host-Side SSD Caching for Storage Performance Control. In *2015 IEEE International Conference on Autonomic Computing* (Grenoble, France, July 7 - 10, 2015) *(ICAC '15)*. IEEE, USA, 51–60. https://doi.org/10.1109/ICAC.2015.44

[29] P.R. Kumar and P. Varaiya. 2015. *Stochastic Systems: Estimation, Identification, and Adaptive Control.* SIAM.

[30] Miryeong Kwon, Donghyun Gouk, Changrim Lee, Byounggeun Kim, Jooyoung Hwang, and Myoungsoo Jung. 2020. DC-Store: Eliminating Noisy Neighbor Containers Using Deterministic I/O Performance and Resource Isolation. In *Proceedings of the 18th USENIX Conference on File and Storage Technologies* (Santa Clara, CA, February 24 - 27, 2020) *(FAST '20)*. USENIX Association, USA, 183–192. https://dl.acm.org/doi/10.5555/3386691.3386709

[31] M. Kwon, J. Zhang, G. Park, W. Choi, D. Donofrio, J. Shalf, M. Kandemir, and M. Jung. 2017. TraceTracker: Hardware/software co-evaluation for large-scale I/O workload reconstruction. In *2017 IEEE International Symposium on Workload Characterization (IISWC)* (Seattle, WA, October 1-3, 2017) *(IISWC '17)*. IEEE Computer Society, USA, 87–96. https://doi.org/10.1109/IISWC.2017.8167759

[32] Eunji Lee, Julie Kim, Hyokyung Bahn, Sunjin Lee, and Sam H. Noh. 2017. Reducing Write Amplification of Flash Storage through Cooperative Data Management with NVM. *ACM Trans. Storage* 13, 2, Article 12 (May 2017). https://doi.org/10.1145/3060146

[33] Cheng Li, Philip Shilane, Fred Douglis, Hyong Shim, Stephen Smaldone, and Grant Wallace. 2014. Nitro: A Capacity-Optimized SSD Cache for Primary Storage. In *Proceedings of the 2014 USENIX Conference on USENIX Annual Technical Conference*

(Philadelphia, PA, June 19 - 20, 2014) *(USENIX ATC '14)*. USENIX Association, USA, 501–512. https://dl.acm.org/doi/10.5555/2643634.2643686

[34] 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 *Proceedings of the 22nd International Conference on Parallel Architectures and Compilation Techniques* (Edinburgh, UK, September 7 - 11, 2013) *(PACT '13)*. IEEE, USA, 103–112. https://doi.org/10.1109/PACT.2013.6618808

[35] Yixin Luo, Yu Cai, Saugata Ghose, Jongmoo Choi, and Onur Mutlu. 2015. WARM: Improving NAND flash memory lifetime with write-hotness aware retention management. In *2015 31st Symposium on Mass Storage Systems and Technologies (MSST)* (Santa Clara, CA, May 30 - June 5, 2015) *(MSST '15)*. IEEE, USA, 1–14. https://doi.org/10.1109/MSST.2015.7208284

[36] Fei Meng, Li Zhou, Xiaosong Ma, Sandeep Uttamchandani, and Deng Liu. 2014. VCacheShare: Automated Server Flash Cache Space Management in a Virtualization Environment. In *Proceedings of the 2014 USENIX Conference on USENIX Annual Technical Conference* (Philadelphia, PA, June 19 - 20, 2014) *(USENIX ATC '14)*. USENIX Association, USA, 133–144. https://dl.acm.org/doi/10.5555/2643634.2643649

[37] Sangwhan Moon and A. L. Narasimha Reddy. 2016. Does RAID Improve Lifetime of SSD Arrays? *ACM Trans. Storage* 12, 3, Article 11 (April 2016). https://doi.org/10.1145/2764915

[38] Muthukumar Murugan and David.H.C. Du. 2011. Rejuvenator: A static wear leveling algorithm for NAND flash memory with minimized overhead. In *2011 IEEE 27th Symposium on Mass Storage Systems and Technologies (MSST)* (Denver, CO, May 23 - 27, 2011) *(MSST '11)*. IEEE, USA, 1–12. https://doi.org/10.1109/MSST.2011.5937225

[39] Yuanjiang Ni, Ji Jiang, Dejun Jiang, Xiaosong Ma, Jin Xiong, and Yuangang Wang. 2016. S-RAC: SSD Friendly Caching for Data Center Workloads. In *Proceedings of the 9th ACM International on Systems and Storage Conference* (Haifa, Israel, June 6 - 8, 2016) *(SYSTOR '16)*. ACM, USA, 1–12. https://doi.org/10.1145/2928275.2928284

[40] Stan Park and Kai Shen. 2012. FIOS: A Fair, Efficient Flash I/O Scheduler. In *Proceedings of the 10th USENIX Conference on File and Storage Technologies* (San Jose, CA, February 14 - 17, 2012) *(FAST '12)*. USENIX Association, USA, 13. https://dl.acm.org/doi/10.5555/2208461.2208474

[41] Liam Parker. 2018. *Optimizing SSDs for Multiple Tenancy Use*. Retrieved October 10, 2022 from https://www.flashmemorysummit.com/English/Collaterals/Proceedings/2018/20180807_SSDS-102-1_Parker.pdf

[42] Craig Partridge. 1994. *Gigabit Networking*. Addison-Wesley Publishing.

[43] Tirthak Patel, Rohan Garg, and Devesh Tiwari. 2020. GIFT: A Coupon Based Throttle-and-Reward Mechanism for Fair and Efficient I/O Bandwidth Management on Parallel Storage Systems. In *Proceedings of the 18th USENIX Conference on File and Storage Technologies* (Santa Clara, CA, February 24 - 27, 2020) *(FAST '20)*. USENIX Association, USA, 103–120. https://dl.acm.org/doi/abs/10.5555/3386691.3386702

[44] Inc. PrimaryIO. 2017. *PrimaryIO APA 2.0 for VMware vSphere ESXi 6.x*. Retrieved October 10, 2022 from https://www.primaryio.com/wp-content/uploads/PrimaryIO-APA2-DataSheet.pdf

[45] Moinuddin K. Qureshi and Yale N. Patt. 2006. Utility-Based Cache Partitioning: A Low-Overhead, High-Performance, Runtime Mechanism to Partition Shared Caches. In *Proceedings of the 39th Annual IEEE/ACM International Symposium on Microarchitecture* (Boston, MA, December 9 - 13, 2006) *(MICRO '06)*. IEEE Computer Society, USA, 423–432. https://doi.org/10.1109/MICRO.2006.49

[46] Kai Shen and Stan Park. 2013. FlashFQ: A Fair Queueing I/O Scheduler for Flash-Based SSDs. In *Proceedings of the 2013 USENIX Conference on Annual Technical Conference* (San Jose, CA, June 26 - 28, 2013) *(USENIX ATC '13)*. USENIX Association, USA, 67–78. https://dl.acm.org/doi/10.5555/2535461.2535471

[47] David Shue, Michael J. Freedman, and Anees Shaikh. 2012. Performance Isolation and Fairness for Multi-Tenant Cloud Storage. In *Proceedings of the 10th USENIX Conference on Operating Systems Design and Implementation* (Hollywood, CA, October 8 - 10, 2012) *(OSDI '12)*. USENIX Association, USA, 349–362. https://dl.acm.org/doi/10.5555/2387880.2387914

[48] Mahesh Subramaniam. 2013. *Exadata Smart Flash Cache Features and the Oracle Exadata Database Machine*. Retrieved October 10, 2022 from https://www.oracle.com/us/solutions/exadata-smart-flash-cache-366203.pdf

[49] Shingo Tanaka. 2018. *Enabling Scalable Ethernet JBOF with a Native NVMe-oF SSD*. Retrieved October 10, 2022 from https://www.flashmemorysummit.com/English/Collaterals/Proceedings/2018/20180809_NVMF-301-1_Tanaka.pdf

[50] Arash Tavakkol, Juan Gómez-Luna, Mohammad Sadrosadati, Saugata Ghose, and Onur Mutlu. 2018. MQsim: A Framework for Enabling Realistic Studies of Modern Multi-Queue SSD Devices. In *Proceedings of the 16th USENIX Conference on File and Storage Technologies* (Oakland, CA, February 12 - 15, 2018) *(FAST '18)*. USENIX Association, USA, 49–65. https://dl.acm.org/doi/10.5555/3189759.3189765

[51] Arash Tavakkol, Mohammad Sadrosadati, Saugata Ghose, Jeremie S. Kim, Yixin Luo, Yaohua Wang, Nika Mansouri Ghiasi, Lois Orosa, Juan Gómez-Luna, and Onur Mutlu. 2018. FLIN: Enabling Fairness and Enhancing Performance in Modern NVMe Solid State Drives. In *Proceedings of the 45th Annual International Symposium on Computer Architecture* (Los Angeles, CA, June 2 - 6, 2018) *(ISCA '18)*. IEEE, USA, 397–410. https://doi.org/10.1109/ISCA.2018.00041

[52] Micron Technology. 2020. *Micron 5300 SSD: Invigorate Your Infrastructure*. Retrieved October 10, 2022 from https://www.micron.com/-/media/client/global/documents/products/product-flyer/5300_product_brief.pdf?la=en

[53] Eno Thereska, Hitesh Ballani, Greg O'Shea, Thomas Karagiannis, Antony Rowstron, Tom Talpey, Richard Black, and Timothy Zhu. 2013. IOFlow: A Software-Defined Storage Architecture. In *Proceedings of the Twenty-Fourth ACM Symposium on Operating Systems Principles* (Farminton, PA, November 3 - 6, 2013) *(SOSP '13)*. ACM, USA, 182–196. https://doi.org/10.1145/2517349.2522723

[54] VMware. 2019. *SSD Endurance for vSAN*. Retrieved October 10, 2022 from https://docs.vmware.com/en/VMware-Validated-Design/5.1/sddc-architecture-and-design/GUID-51680487-239F-4FF7-B43A-8C1D98263DB1.html

[55] Hui Wang and Peter Varman. 2014. Balancing Fairness and Efficiency in Tiered Storage Systems with Bottleneck-Aware Allocation. In *Proceedings of the 12th USENIX Conference on File and Storage Technologies* (Santa Clara, CA, February 17 - 20, 2014) *(FAST '14)*. USENIX Association, USA, 229–242. https://dl.acm.org/doi/abs/10.5555/2591305.2591328

[56] Hui Wang and Peter Varman. 2015. A Resource Allocation Model for Hybrid Storage Systems. In *2015 15th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing* (Shenzhen, China, May 4 - 7, 2015) *(CCGRID '15)*. IEEE, USA, 91–100. https://doi.org/10.1109/CCGrid.2015.132

[57] Hui Wang and Peter Varman. 2016. Time-Based Bandwidth Allocation for Heterogeneous Storage. In *Proceedings of the 2016 ACM SIGMETRICS International Conference on Measurement and Modeling of Computer Science* (Antibes Juanles-Pins, France, June 14 - 18, 2016) *(SIGMETRICS '16)*. ACM, USA, 411–413. https://doi.org/10.1145/2896377.2901492

[58] Wei Wang, Baochun Li, and Ben Liang. 2014. Dominant resource fairness in cloud computing systems with heterogeneous servers. In *IEEE INFOCOM 2014 - IEEE Conference on Computer Communications* (Toronto, ON, Canada, April 27 - May 2, 2014) *(INFOCOM '14)*. IEEE, USA, 583–591. https://doi.org/10.1109/INFOCOM.2014.6847983

[59] Seyed Majid Zahedi and Benjamin C. Lee. 2014. REF: Resource Elasticity Fairness with Sharing Incentives for Multiprocessors. *SIGPLAN Not.* 49, 4 (Feb. 2014), 145–160. https://doi.org/10.1145/2644865.2541962