Fair Resource Allocation in Consolidated Flash Systems

Wonil Choi *Penn State*
Bhuvan Urgaonkar *Penn State*
Mahmut Kandemir *Penn State*
Myoungsoo Jung *KAIST*

Abstract

We argue that, along with bandwidth and capacity, lifetime of flash devices is also a critical resource that needs to be explicitly and carefully managed, especially in emerging consolidated environments. We study the resulting multi-resource allocation problem in a setting where "fairness" across consolidated workloads is desired. Towards this, we propose to adapt the well-known notion of dominant resource fairness (DRF). We empirically show that using DRF with only bandwidth and capacity (and ignoring lifetime) may result in poor device lifetime. Incorporating lifetime, however, turns out to be non-trivial. We identify key challenges in this adaptation and present simple heuristics. We also discuss possible design choices which will be fully explored in future work.

1 Introduction

Increasing storage capacity and bandwidth of flash devices has enabled scenarios wherein multiple workloads are consolidated within a single flash system. Recent work has studied resource management in this emerging flash system [6, 7, 9, 10, 15, 17]. The common design principles identified by this work include (i) the flash system should be workload-aware, (ii) no flash block should be shared among different users, and (iii) resource conflicts among users should be alleviated to provide performance isolation.

While this trend continues, unfortunately, little attention has been paid to "fair" division of flash system resources among participating users. Notions of fairness are often deemed important in private clouds [5] or in public clouds with neutrality-like mandates that may emerge in future [8]. Specifically, we are concerned with the following questions: (i) what are the resources to be divided in consolidated flash systems? and (ii) how one can achieve a fair division of them? For the first question, the resources that come to mind are bandwidth and capacity. However, in addition to these conventional resources, we argue that a flash-specific resource - *flash lifetime* - should be explicitly and carefully managed. Flash cells wear out as they are subjected to writes (and erases) and eventually become unreliable. Flash lifetime has unique characteristics. First, it is non-renewable (consumable) which sets it apart from bandwidth and capacity because perhaps its treatment should change over time (e.g., different closer to a device's end than when it is newly provisioned). Second, it is "consumed" not just by host-induced (explicit or direct) writes but also by implicit requests; specifically, garbage collection (GC) component of flash systems irregularly and unpredictably contributes writes that contribute to lifetime consumption.

As this paper's main focus, we seek to answer the second question raised above when considering all three resources - bandwidth, capacity, and lifetime - together. A recently proposed mechanism for fair division of multiple resources is *Dominant Resource Fairness* (DRF) [5]. Motivated by its fairness properties, we propose to employ DRF in the context of flash systems. Especially, we reveal that flash lifetime needs careful management by comparing DRF that considers all three resources vs. that ignores lifetime.

In managing these three different resources in a coordinated manner, we need to answer the following question: how can one treat lifetime on an equal footing with bandwidth and capacity? Two extreme approaches come to mind. At one extreme, there may be a longer-term goal of ensuring that the device lasts for a certain time (say a year). At another extreme, one may not have any such requirement and be only interested in fairly dividing what lifetime remains at any allocation decision point. For the former, it is natural to spread out available writes across the desired lifetime into allocation "epochs" - periods of relative workload stationarity - where DRF-like allocation is newly performed per epoch with a write "budget" for that epoch. For the latter, since remaining lifetime will keep diminishing with flash usage, a DRF-like allocation will tend to view lifetime as an increasingly bottlenecked resource. Both are plausible scenarios and are partly business decisions, i.e., what kind of behavior do we want – do we want to maintain the illusion of the device acting similarly throughout its life (former) or do we want fewer writes to go to older devices (latter)? More generally, an approach between these two extremes may be appropriate and understanding this is interesting future work. Assuming the former scenario, we make the following contributions:

- We consider flash lifetime as a first-class resource that needs to be managed together with bandwidth and capacity.
- We explore the use of DRF as a multi-resource fair allocation mechanism for consolidated flash systems. We empirically confirm that there is a strong need to incorporate lifetime as an explicit resource in the management of these systems.
- We identify a set of challenges in employing DRF in consolidated flash systems and present preliminary heuristics.

2 Preliminaries

2.1 Dominant Resource Fairness (DRF)

In a computer system with *multiple* resource types, one of the most popular strategies to make a division of the multiple resources is based on Dominant Resource Fairness (DRF) [5]. Observing that each user can have a different dominant re-

source, DRF seeks to equalize dominant shares (i.e., the share that the user has been allocated of dominant resource) across all users; and, this allocation is regarded to be *fair* by satisfying a set of desirable properties [3, 5]. If all users have the same dominant resource, the DRF reduces to max-min fairness (MMF) [2] allocation, which maximizes the minimum allocation received by a user on the dominant resource.

2.2 Our Target System

Stream-based User: One critical requirement in employing DRF is the standardized resource demand of each user. That is, each user has an executable unit (e.g., VM, task, instance) that needs a certain amount of resources; and, the total amount of resources allocated to a user can increase or decrease by launching more or less executable units. Representative examples include database and server applications where the number of clients varies. We refer to the executable unit as *stream*. Note that, while the total resource demand of a user increases linearly with the number of launched streams in vanilla DRF, this is not always the case in reality, which is a challenge (i.e., non-linearity between stream count and total demands) when employing DRF in consolidated flash systems. We will discuss this and present our heuristic later (§5).

Flash Cache: As our target system, we consider a consolidated flash "cache" [1, 11–14] with a backing-store in the next layer, due to two following reasons: (i) allocating limited amount of capacity is a significant task in caching systems (compared to main-storage that should accommodate all of users' data), since the allocated capacity determines hit-ratios and performance of users – the more user capacity a user has, the higher hit-ratio it experiences. (ii) managing (regulating) lifetime is feasible in caching systems; once a user exhausts the allocated amount of writes, its further writes can be prevented by servicing them from the backing-store in the next layer. Note that our proposed resource allocation strategy is agnostic to the exact caching policy; and it is also applicable to main-storage setting.

3 System Resources Considered

We have chosen to follow the conventional resource-based notion of fairness. One reason to pursue such an approach is that it may benefit from some properties [5] that DRF offers. One downside to fairness definitions based on application-specific performance metrics such as hit-ratio and latency is that they open up means for workloads to starve others out.

3.1 Conventional Bandwidth and Capacity

Bandwidth: Across various computing domains (e.g., memory, network), system bandwidth has been one of the representative, prime resource that is divided among multiple users. In particular, (i) how to divide the total bandwidth (e.g., notion of fairness) and (ii) how to guarantee the partitioned bandwidth (e.g., performance isolation) are two major concerns addressed in the literature. For consolidated flash systems, recent studies [6, 9, 10, 16] demonstrated that provisioning a specified bandwidth for each user in them is feasible.

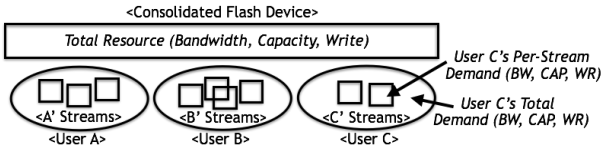

Figure 1: Illustration of our allocation problem: a flash device's 3 types of resources are to be divided across competing users, each with its unique per-stream resource demand.

Capacity: In memory and storage systems, device capacity also has been a key resource to be partitioned among users. Particularly, any caching device where all user data cannot be accommodated needs to determine how much capacity is allocated to a user, which directly affects the hit ratio and the performance of the user. Considering various placement of flash devices in the memory-storage hierarchy, various studies have focused on flash capacity partitioning [1, 11–14].

3.2 Flash-Specific Device Lifetime

We pay attention to another significant, flash-specific resource, namely, flash lifetime. As opposed to the other conventional resources, lifetime (or, its proxy, write count) has two unique characteristics. First, it is consumable (non-renewable); in other words, a flash memory can process only a limited number of write operations, beyond which the device becomes unavailable. Since the limited lifetime resource has a direct relationship with the ownership cost (economical condition), it can be considered as the first-class resource to be managed in some cases. Second, the lifetime is affected by other resources. Specifically, the write count is consumed by both explicit write requests (from the host) and implicit write traffic (from the device-internals); and the latter is significantly affected by the storage capacity assigned to it. That is, the number of writes issued by the garbage collection (GC) is determined by the over-provisioned (OP) capacity.

There is no uniform definition of fairness in device wear-out. We are interested in a "multi-resource" fairness definition where lifetime/writes is one resource out of many. If one insists on thinking about wear-out alone, it may help to imagine infinite capacity and bandwidth but finite lifetime. The problem would then boil down to MMF allocation [2] of lifetime.

3.3 Coordination of Three Resources

To treat the three different resource in an orchestrated manner, we define "epoch" (a period of relative workload stationarity) and, in every epoch, our DRF framework newly divides the given total resources to the given competing users. In an epoch, each user and their streams are assumed to be backlogged – each continues to consume a fixed amount of bandwidth while taking a fixed amount of storage capacity. Regarding the lifetime, on the other hand, a total write budget is prescribed to the epoch; and all users and their streams are supposed to share (and divide) the given write budget. In this epoch-based framework, we can manage the three resources on relatively different time scale in a coordinated fashion.

Device's Total Resources for 1h Epoch	Total Bandwidth (BW): 500 MB/s; Total Capacity (CAP): 256 GB; Total Write Budget (WR): 11,184,810; 1TB/4K-PAGE/24h				
User's Per-Stream Demands	# Host I/O		Data Size (GB)		Amp.
	WR	RD	Shared	Per-Stream	Value
A	1,000,000	100,000	20	5	3.0
B	100,000	1,000,000	60	20	1.2
C	300,000	50,000,000	30	10	1.5

Table 1: Configuration of device and workloads: an example epoch where 3 types of resources are divided by 3 users.

4 DRF Allocation Examples

Figure 1 illustrates our target problem. A flash system provides three types of resources: bandwidth, capacity, and lifetime. In a given epoch, these resources are allocated to multiple users, each of which has its unique resource demand. We assume that each user's bandwidth, capacity, and lifetime demands (resource usages in the worst-case scenario) can be obtained using static workload profiling and analytical models. For example, the total lifetime demand of a user can be calculated by considering (i) # host writes it generates, (ii) its own write amplification (WA) model, and (iii) over-provisioned (OP) capacity allocated to it. We implemented our framework using MATLAB and explored how the resource allocation changes depending on the inputs (device's resources and users' demand).

4.1 DRF Considering All Three Resources

Example: Let us consider an example scenario to see how DRF allocates the three resources among the three participants. Table 1 lists up the device's total resources and the per-stream characteristics (e.g., host I/O, data size, write amplification) of the three users. User A mainly needs the write resource as its stream generates many host writes and exhibits a high write amplification (WA) value. User B dominantly needs the capacity resource as its stream loads a large amount of data into device. Finally, User C primarily wants the bandwidth resource as its stream generates intensive host reads.

Allocation Process: Let us visit how DRF allocates the three resources. Algorithm 1 describes the DRF algorithm. At first, (i) it randomly picks a user, launches its stream, and counts the allocated resource. Next, (ii) it finds the dominant share (s_i) of each user – the dominant share is the maximum fraction of allocated amount to the total amount among the three resources. Then, (iii) it finds the user who has the minimum dominant share; and, the user acquires more resources by launching its stream if there are still available resource to do so. Here, (iv) actual resource demands (b_i, c_i, w_i) in launching the additional stream should be estimated from per-stream characteristics. We present our simple heuristic in §5. Finally, it repeats (ii)∼(iv) until no more streams can be launched due to lack of (remaining) resources.

Result: Both Table 2 and Figure 2 show how DRF allocates the three resources among the three users. Specifically, DRF tries to equalize the dominant shares of all users while increasing the number of launched streams. As indicated in Table 2, the dominant shares of Users A, B, and C are close to each other, whereas their dominant resources are all different. As a result, Users A, B, and C are allocated the specified amount of resources by launching 2, 4, and 5 streams, respectively. Figure 2 also shows that the largest areas of the users across the resources (write allocation of A, capacity allocation of B, and bandwidth allocation of C) are nearly equal. Note that, the bandwidth and write resources are not fully allocated, since the capacity resource is almost exhausted and the remaining capacity is not enough to launch a stream from any user.

Algorithm 1: DRF allocation of three resources.

Input: Device's total resources ($< B, C, W >$, where B, C, and W are bandwidth, capacity, and writes, respectively), the number of users (N), users' capacity demands ($\hat{c}$).
Output: Users' allocated resources ($A_i =< b_i, c_i, w_i >$, where b_i, c_i, and w_i are bandwidth, capacity, and writes allocated to user i), the numbers of launched streams (k_i, where the number of streams for user i).

$< CB, CC, CW > \leftarrow < 0, 0, 0 >$ // Device's consumed resources
$< DB_i, DC_i, DW_i >$ // User i's demands for the next stream
s_i // User i's dominant share
Pick a random user i
$DC_i \leftarrow \hat{c}_i(1)$, $DW_i \leftarrow \hat{w}_i(1)$, $DB_i \leftarrow \hat{b}_i(1)$
while $(CB + DB_i \leq B) \& (CC + DC_i \leq C) \& (CW + DW_i \leq W)$ **do**
  Launch a stream for user i
  // Update # of launched streams
  $k_i \leftarrow k_i + 1$
  // Update allocated resources A_i
  $b_i \leftarrow b_i + DB_i, c_i \leftarrow c_i + DC_i, w_i \leftarrow w_i + DW_i$
  // Update device's consumed resources
  $CB \leftarrow CB + DB_i, CC \leftarrow CC + DC_i, CW \leftarrow CW + DW_i$
  // Find dominant share of each user
  for $i \leftarrow$ *all participating users* **do**
    $s_i \leftarrow \max(b_i/B, c_i/C, w_i/W)$
  end
  // Find the user with the lowest dominant share
  $i \leftarrow \min_{j=1}^{N} s_i$
  // Estimate user i's demands for the next stream
  $DC_i \leftarrow \hat{c}_i(k_i + 1) - \hat{c}_i(k_i)$
  $DW_i \leftarrow \hat{w}_i(k_i + 1) - \hat{w}_i(k_i)$
  $DB_i \leftarrow \hat{b}_i(k_i + 1) - \hat{b}_i(k_i)$
end

Other Scenarios: The three users in the above example are representative of various workload types. That is, each of three is bandwidth-dominant, capacity-dominant, or lifetime-dominant. We examined many other user (consolidation) scenarios, whose results are left out, due to space limit. One general observation to highlight is that, when consolidated users are similar (i.e., all are lifetime-dominant or bandwidth-dominant), the DRF allocation is equal to the MMF [2] allocation of the common dominant resource.

4.2 DRF without Lifetime Management

2-Resource DRF: Considering non-renewable lifetime as a precious resource that should be carefully and explicitly managed, we included it the DRF allocation above. To see the impact of ignoring lifetime management, we employ a 2-resource DRF where only the two conventional resources (bandwidth and capacity) are considered in the same example (Table 1). The process of equalizing dominant shares of all users does not change; but, the dominant share (s_i) is calculated by $\max(b_i/B, c_i/C)$ since lifetime is out of the picture.

Allocation Result: Table 3 and Figure 3 show how 2-resource DRF allocates the two resources among the three users. One important point is that User A's dominant resource changes

User	Dom. Share	Dom. Share (Alloc./Total)	# streams	Alloc. BW	Alloc. CAP	Alloc. WR
A	Write	6,000,000/11,184,810 = **0.536**	**2**	7 MB/s	30 GB	**6,000,000**
B	Capacity	140GB/256GB = **0.546**	**4**	5 MB/s	**140 GB**	480,000
C	Bandwidth	274MBs/512MBs = **0.535**	**5**	**274 MB/s**	80 GB	2,250,000
			Alloc. Sum	286 MB/s	250 GB	8,730,000
			Total	512 MB/s	256 GB	11,184,810

Table 2: 3-resource DRF: it tries to equalize the dominant shares of all three users.

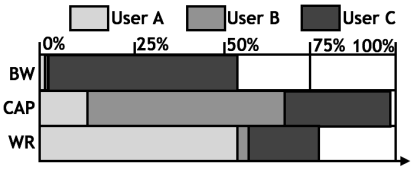

Figure 2: 3-resource DRF allocation.

User	Dom. Share	Dom. Share (Alloc./Total)	# streams	Alloc. BW	Alloc. CAP	Used WR
A	**Capacity**	85GB/256GB = **0.332**	**13**	44 MB/s	**85 GB**	39,000,000
B	Capacity	100GB/256GB = **0.391**	**2**	3 MB/s	**100 GB**	240,000
C	Bandwidth	219MBs/512MBs = **0.427**	**4**	**219 MB/s**	70 GB	1,800,000
WR value indicates the sum of used writes.			**Alloc. Sum**	266 MB/s	255 GB	**41,040,000**
WR value comes from total WR resource of the baseline.			**Total**	512 MB/s	256 GB	(11,184,810)

Table 3: 2-resource DRF (where lifetime is not considered): it ends up in an entirely new allocation compared to 3-resource DRF, and this causes a significant write consumption.

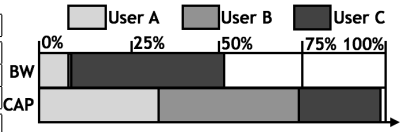

Figure 3: 2-Resource DRF's allocation. While bandwidth and capacity are considered, lifetime is not managed.

from *write* in 3-resource DRF to *capacity*. As usual, the DRF tries to equalize the dominant shares while maximizing the number of launched streams. As a result, Users A, B, and C are allocated the specified amount of resources by launching 13, 2, and 4 streams, respectively, which is *significantly different* from the 3-resource DRF allocation. Since User A's capacity demand is very low and its dominant share also remains low, a lot of streams are launched. Figure 3 also shows that the largest areas of the users across the resources (capacity allocation of A, capacity allocation of B, and bandwidth allocation of C) are fairly close to each other.

Lifetime Implication: In the case of 2-resource DRF allocation, the expected number of writes to be used by the three users is 41,040,000, which is more than 3× of the write budget of the baseline. Specifically, 95% of the writes are used by User A, since User A is allowed to launch up to 13 streams and; these streams collectively consume a considerable amount of writes. One can conclude from this observation that, if flash lifetime is not carefully managed, (i) while bandwidth and capacity are "fairly" divided to users, their write consumption does not look fair; and, (ii) overall, the end of flash life can be accelerated. Therefore, if one wants his/her flash device to last longer, the DRF-based resource allocation should take the writes (lifetime) into account. Note however that, in this scenario (if device operator does not manage the lifetime), the other resources (bandwidth and capacity) provided by the device can be fairly (or almost fully) allocated to the competing users in our framework.

5 Challenges: User's Demand Estimation

To adapt the DRF, the following question needs to be addressed: if an additional stream is launched for a user, how much bandwidth, capacity, and write resource does the user ask? That is, one needs to obtain the actual resource demands for each user with a given number of streams. However, this is not a straightforward task, due to the following challenges.

Non-Linearity in Demand Increase: If a resource demand increases linearly with streams, estimating the total demand is easy. In fact, this can be done for the bandwidth and write demands of a user, since each stream has its own host I/O and GC traffic. Unfortunately, capacity demand is different, since streams of a user commonly share part of the data. A typical example is a database application where frequently-accessed data are cached and its multiple clients share the data.

Let us see how DRF works if users' capacity demands lack the knowledge of shared data and increases linearly with streams. We modify the values of "shared data size" and "per-stream data size" of the three users in our baseline Table 1; shared data sizes are assumed to be zero while per-stream data sizes are a bit adjusted (Table 4). In this scenario, User C's dominant resource changes from bandwidth to capacity, as its capacity demand significantly (linearly) increases with streams. As a result, DRF makes an entirely-new allocation against the baseline (Table 2), where Users A, B, and C launch 2, 1, and 2 streams, respectively. Actually, this is not a desirable scenario, since resources (bandwidth and write) are significantly under-utilized. Specifically, the total allocated bandwidth and write decrease from 286MB/s and 8,730,000 to 174MB/s and 7,470,000, respectively. This is because the linear increase in capacity demands makes the capacity resource fully allocated with a small number of streams, and as a result, other resources lose opportunity to be further allocated.

To this end, we propose a simple heuristic to construct the capacity demand by being aware of the shared data among streams. User i's capacity demand with k streams ($c_i(k)$) consists of a fixed capacity for share data (c_i^s) and a set of capacities for per-stream data (c_i^p).

$$\hat{c}_i(k) = c_i^s + c_i^p \times k. \quad (1)$$

Note that the allocation examples so far are based on above heuristic, from which one can conclude that DRF can make a better resource allocation once the capacity demands (non-linearity with stream) of each user are accurately estimated. Note also that, the scenario of inter-user data sharing (data are shared across users) will be factored in future work.

Correlation among Resource Demands: Resource demands of a user are correlated in complex ways; consequently, estimation of total demands of a user should take this correlation into account. For example, let us assume that a user

User	Shared Data	Per-Stream Data	Dom. Share	Dom. Share (Alloc./Total)	# streams	Alloc. BW	Alloc. CAP	Alloc. WR
A	**0 GB**	25 GB	Write	6,000,000/11,184,810 = **0.536**	**2**	8 MB/s	50 GB	**6,000,000**
B	**0 GB**	80 GB	Capacity	80GB/256GB = **0.313**	**1**	2 MB/s	**80 GB**	120,000
C	**0 GB**	40 GB	**Capacity**	120GB/256GB = **0.468**	**2**	164 MB/s	**120 GB**	1,350,000
Note that streams have no shared data!					**Alloc. Sum**	174 MB/s	250 GB	7,470,000
So, capacity demands increase linearly with streams.					**Total**	512 MB/s	256 GB	11,184,810

Table 4: 3-resource DRF which is unaware of non-linearity increase in capacity demands and assumes zero shared data sizes.

User	Per-Stream Data	Amp. Value	Dom. Share	Dom. Share (Alloc./Total)	# streams	Alloc. BW	Alloc. CAP	Alloc. WR
A	**10 GB**	**1.5**	Write	4,500,000/11,184,810 = **0.402**	**3**	6 MB/s	50 GB	**4,500,000**
B	**10 GB**	**2**	Capacity	110GB/256GB = **0.429**	**5**	7 MB/s	**110 GB**	1,000,000
C	**15 GB**	**1.2**	Bandwidth	219MBs/512MBs = **0.427**	**4**	**219 MB/s**	90 GB	1,440,000
Note that per-stream data size and amp. value change from the baseline.					**Alloc. Sum**	232 MB/s	250 GB	6,940,000
So, changes in capacity demands lead to changes in write demands.					**Total**	512 MB/s	256 GB	11,184,810

Table 5: 3-resource DRF where the capacity demand (OP size) and the corresponding write (WA value) demand are changed.

expects (needs) a high hit ratio from a large capacity; then, its bandwidth demand should correspondingly increase to accommodate the high hit ratio. Among a range of relationships, we focus on an interesting one between capacity demand and write demand. In general, the more (OP) capacity an application has, the less GC overheads (GC writes) it experiences [4]. Here, let us assume that the OP capacity of a user is part of its per-stream capacity (§6). Accordingly, if a user increases its capacity demand (per-stream data size), its write demand should also decrease correspondingly, since its WA value and GC writes decrease. Keeping this in mind, we propose a simple heuristic to estimate the write demand of a user, given its capacity demand. Specifically, User i's write demand with k streams ($w_i(k)$) can be calculated as follows:

$$\hat{w}_i(k) = k \times w_i^h(1) \times A_i(c_i^o(k)), \quad (2)$$

where A_i and c_i^o are the amplification function and the OP capacity of user i. For the amplification function, one can employ an existing analytic model [4]. Our simple heuristic is based on the assumptions that (i) the number of host writes (w^h) linearly increases with the number of streams (i.e., $w_i^h(k) = k \times w_i^h(1)$), and (ii) streams have the same write amplification value since their host write traffic and OP capacity are assumed to be the same.

We examine another scenario where both capacity and write demands are adjusted. Note that, "per-stream data size" (OP in it) and "amp. value" are adjusted in Table 5. Under the changed capacity and write demands, the DRF leads to a new allocation against the baseline. Specifically, Users A, B, and C are allowed to launch 3, 5, and 4 streams, respectively. As this example shows, one can construct the most beneficial resource demand vector for a user by exploring the complex correlation among resource demands and adjusting their amounts.

6 DRF Design Parameters

OP Fraction of Allocated Capacity: One can raise a question: how much OP capacity is given to a user out of the total allocated capacity? We use the following simple heuristic to specify the amount of the OP capacity. The OP capacity of User i with k streams ($c_i^o(k)$) can be calculated as follows:

$$c_i^o(k) = k \times c_i^p \times f, \quad (3)$$

where c^p is per-stream capacity and f is user-specified fraction of OP capacity to the entire per-stream capacity. It is critical to find the most appropriate f value, since it determines both the capacity and write demands, which can collectively and significantly affect the resulting DRF allocation.

Lifetime Management Policy: In our baseline example, the total write resource 11,184,810 (Table 1) represents the number of 4KB-page writes allowed for a period of 1 hour, assuming that only 1TB write consumption is permitted for a day. However, there can be a range of options for managing a device's lifetime; for example, one might set a tighter budget to last his/her device much longer, while one might not concern the device wear-out and the replacement cost. To see the impact of lifetime management policy, we consider an extreme scenario where the total write resource is set to all the remaining writes to the end of device's life. According to our analysis, when we set the total write resource to a very large number (assuming that our device is young), the DRF results in the same allocation as the 2-resource DRF case where the lifetime resource is not managed (Table 3). This is because the large write resource makes User A's dominant resource the *capacity* (instead of *write*) and DRF excludes the capacity resource from its interest. However, the situation can change again as the device gets older and its remaining writes becomes small. We will explore the lifetime management choices to best allocate multiple resources in future work.

7 Conclusions

DRF is a useful tool to divide multiple resource types across competing users. We propose to employ DRF in consolidated flash systems where bandwidth, capacity, and lifetime resources are correlated in complex ways. We explore challenges in DRF adaptation and design parameters that can affect the resource allocation. In future work, we will study the notion of lifetime fairness in multiple device settings.

Acknowledgments

We thank Anirudh Badam, 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. Jung's research is in part supported by NRF 2016R1C1B2015312, DOE DEAC02-05CH11231, NRF 2015M3C4A7065645, NRF 2017R1A4A1015498 and Samsung grant (IO181008-05622-01).

References

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

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

[3] Mosharaf Chowdhury, Zhenhua Liu, Ali Ghodsi, and Ion Sotica. HUG: Multi-Resource Fairness for Correlated and Elastic Demands. In *NSDI*, 2016.

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

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

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

[7] Jeong-Uk Kang, Jeeseok Hyun, Hyunjoo Maeng, and Sangyeun Cho. The Multi-streamed Solid-State Drive. In *USENIX HotStorage*, 2014.

[8] George Kesidis, Bhuvan Urgaonkar, Neda Nasiriani, and Cheng Wang. Neutrality in Future Public Clouds: Implications and Challenges. In *USENIX HotCloud*, 2016.

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

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

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

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

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

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

[15] Eunhee Rho, Kanchan Joshi, Seung-Uk Shin, Nitesh Jagadeesh Shetty, Jooyoung Hwang, Sangyeun Cho, Daniel DG Lee, and Jaeheon Jeong. FStream: Managing Flash Streams in the File System. In *USENIX FAST*, 2018.

[16] Hui Wang and Peter Varman. Balancing Fairness and Efficiency in Tiered Storage Systems with Bottleneck-Aware Allocation. In *USENIX FAST*, 2014.

[17] Jingpei Yang, Rajinikanth Pandurangan, Changho Choi, and Vijay Balakrishnan. AutoStream: Automatic Stream Management for Multi-streamed SSDs. In *SYSTOR*, 2017.